Go Grammar

Read carefully:

1- This grammar is a modified fragment of the Go language specifications .

2- According to the Go language specifications, these are the following conventions used for stating the grammar:

- [] denoates 0 or 1.
- {} denoates 0 or more .
- () is used for grouping one or more expression together.

3- All **BOLD** literals are the tokens you defined in the lexer. You are only allowed to change in how the operators tokens are defined.

NOTE:

The fragment introduced in milestone 1 was not an LALR grammar. So to be able to do it with CUP some modifications were made to make it easier for you.

The Modified Grammar

binary_op = **"||"** | **"&&"** | **rel_op** | add_op | mul_op .

add_op = **"+"** | **"-"** | **"|"** .

mul_op = **"*"** | **"/"** | **"%"** | **"<<"** | **">>"** | **"&"** .

unary_op = **"+"** | **"-"** | **"!"** | **"*"** | **"&"** | **"<-"** .

Type = TypeName | TypeLit .

TypeName = **identifier** .

TypeLit = ArrayType | StructType | FunctionType | SliceType

ArrayType = **"["** ArrayLength **"]"** ElementType .

ArrayLength = Expression .

ElementType = Type .

SliceType = **"["** **"]"** ElementType .

StructType = **"struct"** **"{"** { FieldDecl **";"** } **"}"** .

FieldDecl = (IdentifierList Type | AnonymousField) [Tag] .

AnonymousField = [**"*"**] TypeName .

Tag = **string_lit** .

FunctionType = **"func"** Signature .

Signature = Parameters [Result] .

Result = Parameters | **"("** Type **")"** .

Parameters = **"("** [ParameterList] **")"** .

ParameterList = ParameterDecl { **","** ParameterDecl } .

ParameterDecl = IdentifierList [**"..."**] Type .

MethodName = **identifier** .

Block = **"{"** StatementList **"}"** .

StatementList = { Statement } .

Declaration = ConstDecl **";"** | TypeDecl [**";"**] | VarDecl **";"** .

TopLevelDecl = Declaration | FunctionDecl [**";"**] | MethodDecl [**";"**].

ConstDecl = **"const"** (ConstSpec | **"("** { ConstSpec **";"** } **")"**) .

ConstSpec = **identifier** [[Type] **"="** Expression] .

IdentifierList = **identifier** { **","** **identifier** } .

ExpressionList = Expression { **","** Expression } .

Expression = UnaryExpr | Expression binary_op Expression .

UnaryExpr = PrimaryExpr | unary_op UnaryExpr .

TypeDecl = **"type"** (TypeSpec | **"("** { TypeSpec **";"** } **")"**) .

TypeSpec = **identifier** Type .

VarDecl = **"var"** (VarSpec | **"("** { VarSpec **";"** } **")"**) .

VarSpec = **identifier** (Type [**"="** Expression] | **"="** Expression) .

ShortVarDecl = IdentifierList **":="** Expression .

FunctionDecl = **"func"** FunctionName (Function | Signature) .

FunctionName= **identifier** .

Function = Signature FunctionBody .

FunctionBody = Block .

MethodDecl = **"func"** Receiver MethodName (Function | Signature) .

Receiver = Parameters .

Operand = Literal | OperandName | MethodExpr | **"("** Expression **")"** .

Literal = BasicLit | CompositeLit | FunctionLit .

CompositeLit = LiteralType LiteralValue .

LiteralType = StructType | ArrayType | **"["** **"..."** **"]"** ElementType | SliceType | **"type"** TypeName .

LiteralValue = **"{"** [ElementList] **"}"** .

ElementList = KeyedElement { **","** KeyedElement } .

KeyedElement = [Key **":"**] Element .

Key = FieldName | LiteralValue .

FieldName = identifier .

Element = Expression | LiteralValue .

BasicLit = **int_lit** | **string_lit** .

OperandName = **identifier** | QualifiedIdent.

QualifiedIdent = **"."** PackageName **"."** **identifier** .

FunctionLit = **"func"** Function .

PrimaryExpr = Operand | PrimaryExpr Selector | PrimaryExpr Index | PrimaryExpr Slice
| PrimaryExpr Arguments .

Selector = **"."** **identifier** .

Index = **"["** Expression **"]"** .

Slice = **"["** [Expression] **":"** [Expression] **"]"** | **"["** [Expression] **":"** Expression **":"** Expression **"]"** .

Arguments = **"("** [[**"type"** Type **","**] ExpressionList] **")"** .

MethodExpr = **"."** ReceiverType **"."** MethodName .

ReceiverType = **"("** **"*"** TypeName **")"** | **"("** TypeName **")"** .

Statement = Declaration | SimpleStmt **";"** | ReturnStmt **";"** | BreakStmt **";"** | Block [**";"**] | IfStmt [**";"**]
| SwitchStmt [**";"** **]** | ForStmt [**";"**] .

SimpleStmt = ExpressionStmt | IncDecStmt | Assignment | ShortVarDecl .

ExpressionStmt = Expression .

IncDecStmt = Expression (**"++"** | **"--"**) .

Assignment = ExpressionList assign_op ExpressionList .

assign_op = [add_op | mul_op] **"="** .

IfStmt = **"if"** [SimpleStmt **";"**] Expression Block [**"else"** (IfStmt | Block)] .

SwitchStmt = ExprSwitchStmt .

ExprSwitchStmt = **"switch"** [SimpleStmt **";"**] [Expression] **"{"** { ExprCaseClause } **"}"** .

ExprCaseClause = ExprSwitchCase **":"** StatementList .

ExprSwitchCase = **"case"** ExpressionList | **"default"** .

ForStmt = **"for"** [Condition | ForClause] Block .

Condition = Expression .

ForClause = [InitStmt] **";"** [Condition] **";"** [PostStmt] .

InitStmt = SimpleStmt .

PostStmt = SimpleStmt .

ReturnStmt = **"return"** [ExpressionList] .

BreakStmt = **"break"** .

SourceFile = PackageClause [**";"**] { ImportDecl [**";"**] } { TopLevelDecl } .

PackageClause = **"package"** PackageName .

PackageName = **identifier** .

ImportDecl = **"import"** (ImportSpec | **"("** { ImportSpec [**";"**] } **")"**) .

ImportSpec = [**"."** | PackageName] ImportPath .

ImportPath = **string_lit** .