# Syntax of the Goo Language Subset

The grammar rules below are copied verbatim from the on-line document "The Go Programming Language Specification" (`https://golang.org/ref/spec`), preserving the same order. No rules for lexical structure are included.

Rules which are coloured black are in the Goo subset for Assignment 1. **Rules and parts of rules which are coloured red are NOT in the Goo subset for Assignment 1**. (Not all these Goo language features will necessarily be supported in Assignments 2 to 4, but their syntax will be checked.)

For the meanings of these constructs, please refer to the Go language specification.

```
Type     = TypeName | TypeLit | "(" Type ")" .
TypeName = identifier | QualifiedIdent .
TypeLit  = ArrayType | StructType | PointerType | FunctionType | InterfaceType |
           SliceType | MapType | ChannelType .

ArrayType   = "[" ArrayLength "]" ElementType .
ArrayLength = Expression .
ElementType = Type .

SliceType = "[" "]" ElementType .

StructType     = "struct" "{" { FieldDecl ";" } "}" .
FieldDecl      = (IdentifierList Type | AnonymousField) [ Tag ] .
AnonymousField = [ "*" ] TypeName .
Tag            = string_lit .

PointerType = "*" BaseType .
BaseType    = Type .

FunctionType   = "func" Signature .
Signature      = Parameters [ Result ] .
Result         = Parameters | Type .
Parameters     = "(" [ ParameterList [ "," ] ] ")" .
ParameterList  = ParameterDecl { "," ParameterDecl } .
ParameterDecl  = [ IdentifierList ] [ "..." ] Type .

InterfaceType      = "interface" "{" { MethodSpec ";" } "}" .
MethodSpec         = MethodName Signature | InterfaceTypeName .
MethodName         = identifier .
InterfaceTypeName  = TypeName .

MapType  = "map" "[" KeyType "]" ElementType .
KeyType  = Type .

ChannelType = ( "chan" | "chan" "<-" | "<-" "chan" ) ElementType .

Block = "{" StatementList "}" .
StatementList = { Statement ";" } .

Declaration   = ConstDecl | TypeDecl | VarDecl .
```

```
TopLevelDecl  = Declaration | FunctionDecl | MethodDecl .

ConstDecl = "const" ( ConstSpec | "(" { ConstSpec ";" } ")" ) .
ConstSpec = IdentifierList [ [ Type ] "=" ExpressionList ] .

IdentifierList = identifier { "," identifier } .
ExpressionList = Expression { "," Expression } .

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

VarDecl = "var" ( VarSpec | "(" { VarSpec ";" } ")" ) .
VarSpec = IdentifierList ( Type [ "=" ExpressionList ] | "=" ExpressionList ) .

ShortVarDecl = IdentifierList ":=" ExpressionList .

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 .
BasicLit    = int_lit | float_lit | imaginary_lit | rune_lit | string_lit .
OperandName = identifier | QualifiedIdent.

QualifiedIdent = PackageName "." identifier .

CompositeLit  = LiteralType LiteralValue .
LiteralType   = StructType | ArrayType | "[" "..." "]" ElementType |
                SliceType | MapType | TypeName .
LiteralValue  = "{" [ ElementList [ "," ] ] "}" .
ElementList   = Element { "," Element } .
Element       = [ Key ":" ] Value .
Key           = FieldName | Expression | LiteralValue .
FieldName     = identifier .
Value         = Expression | LiteralValue .

FunctionLit = "func" Function .

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

Selector  = "." identifier .
Index     = "[" Expression "]" .
```

```
Slice     = "[" ( [ Expression ] ":" [ Expression ] ) |
                ( [ Expression ] ":" Expression ":" Expression )
                 "]" .
TypeAssertion  = "." "(" Type ")" .
Arguments   = "(" [ ( ExpressionList | Type [ "," ExpressionList ] )]
                  [ "..." ] [ "," ] ] ")" .

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

Expression = UnaryExpr | Expression binary_op Expression .
UnaryExpr  = PrimaryExpr | unary_op UnaryExpr .

binary_op  = "||" | "&&" | rel_op | add_op | mul_op .
rel_op     = "==" | "!=" | "<" | "<=" | ">" | ">=" .
add_op     = "+" | "-" | "|" | "^" .
mul_op     = "*" | "/" | "%" | "<<" | ">>" | "&" | "&^" .

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

Conversion = Type "(" Expression [ "," ] ")" .

Statement =
   Declaration | LabeledStmt | SimpleStmt |
   GoStmt | ReturnStmt | BreakStmt | ContinueStmt | GotoStmt |
   FallthroughStmt | Block | IfStmt | SwitchStmt | SelectStmt | ForStmt |
   DeferStmt .

SimpleStmt = EmptyStmt | ExpressionStmt | SendStmt | IncDecStmt |
             Assignment | ShortVarDecl .

EmptyStmt = .

LabeledStmt = Label ":" Statement .
Label       = identifier .

ExpressionStmt = Expression .

SendStmt = Channel "<-" Expression .
Channel  = Expression .

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

Assignment = ExpressionList assign_op ExpressionList .

assign_op = [ add_op | mul_op ] "=" .

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

SwitchStmt = ExprSwitchStmt | TypeSwitchStmt .

ExprSwitchStmt = "switch" [ SimpleStmt ";" ] [ Expression ]
                 "{" { ExprCaseClause } "}" .
ExprCaseClause = ExprSwitchCase ":" StatementList .
```

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

TypeSwitchStmt  = "switch" [ SimpleStmt ";" ] TypeSwitchGuard
                  "{" { TypeCaseClause } "}" .
TypeSwitchGuard = [ identifier ":=" ] PrimaryExpr "." "(" "type" ")" .
TypeCaseClause  = TypeSwitchCase ":" StatementList .
TypeSwitchCase  = "case" TypeList | "default" .
TypeList        = Type { "," Type } .

ForStmt = "for" [ Condition | ForClause | RangeClause ] Block .
Condition = Expression .

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

RangeClause = [ ExpressionList "=" | IdentifierList ":=" ] "range" Expression .

GoStmt = "go" Expression .

SelectStmt = "select" "{" { CommClause } "}" .
CommClause = CommCase ":" StatementList .
CommCase   = "case" ( SendStmt | RecvStmt ) | "default" .
RecvStmt   = [ ExpressionList "=" | IdentifierList ":=" ] RecvExpr .
RecvExpr   = Expression .

ReturnStmt = "return" [ ExpressionList ] .

BreakStmt = "break" [ Label ] .

ContinueStmt = "continue" [ Label ] .

GotoStmt = "goto" Label .

FallthroughStmt = "fallthrough" .

DeferStmt = "defer" Expression .

SourceFile = PackageClause ";" { ImportDecl ";" } { TopLevelDecl ";" } .

PackageClause  = "package" PackageName .
PackageName    = identifier .

ImportDecl  = "import" ( ImportSpec | "(" { ImportSpec ";" } ")" ) .
ImportSpec  = [ "." | PackageName ] ImportPath .
ImportPath  = string_lit .
```