Algorithm for SNR Parser

Mathematical Representation of `global_scope`

1. **Initialization**
    - Let **TokenStream** = $\{t_1, t_2, ..., t_n\}$, where t_i is the i-th token in the input program.
    - Define parsing states:
      S = {IsMultipleDecl, IsFunctionDef, IsFunctionDecl, IsBlockEnd, IsEnumDef, IsEnumFunctionDef,
      IsEnumFunctionDecl, IsStructDef, IsStructDecl, IsStructFunctionDef,
      IsStructFunctionDecl, IsFunctionCaller, IsStatement}.
    - Initialize $P(s) \leftarrow FALSE \ \forall \ s \in S$.
    - Initialize $startToken(s) \leftarrow 0 \ \forall \ s \in S$.
2. **Grammar Check Function**
    - The Grammar Check function determines whether the input token t_i matches the grammar rules for a given state.
    - For each parsing state, the grammar rules are evaluated against the input token to determine correctness.
    - Specific rules include:
        - MultipleDeclaration(t_i) (M): checks for multiple declarations.
        - FunctionDefinitionOrDeclaration(t_i) (F): verifies function definitions or declarations.
        - BlockEndStatement(t_i) (B): checks for the end of a block.
        - EnumDefinition(t_i) (E): evaluates enum definitions.
        - StructDefinitionOrDeclaration(t_i) (S): identifies struct-related rules.
        - FunctionCaller(t_i) (C): checks for function calls.
        - Statement(t_i) (St): identifies statements.
3. **Parsing Workflow**
    1. For each $t_i \in$ TokenStream:
        1. Record the Current Token Position:
           $startToken(s) \leftarrow i$ (where t_i is the current token) $\forall \ s \in S$.
        2. Parse Language Constructs:
            - $P(IsMultipleDecl) \leftarrow M(t_i)$
            - $P(IsFunctionDef), P(IsFunctionDecl) \leftarrow F(t_i)$
            - $P(IsBlockEnd) \leftarrow B(t_i)$
            - $P(IsEnumDef), P(IsEnumFunctionDef), P(IsEnumFunctionDecl) \leftarrow E(t_i)$

        - P(IsStructDef), P(IsStructDecl), P(IsStructFunctionDef), P(IsStructFunctionDecl) ← $S(t_i)$
        - P(IsFunctionCaller) ← $C(t_i)$
        - P(IsStatement) ← $St(t_i)$.
4. **Compute Distance**
    - For each successful parsing state s:
        - dist(s) ← i - startToken(s).
        - Report the corresponding construct:
          - "multiple declaration" if P(IsMultipleDecl) ← TRUE.
          - "function definition" if P(IsFunctionDef) ← TRUE.
          - "function declaration" if P(IsFunctionDecl) ← TRUE.
          - ... for other states.
5. **Error Handling**
    - If no parsing state is successful:
        - ShowError().
        - Terminate the process.
6. **Reinitialize Parsing Table**
    - Reset P(s) ← FALSE ∀ s ∈ S for the next token.
7. **Exit Condition**
    - The loop ends when t_i = EOF.
    - If all tokens are processed without errors, the parsing is successful.