# SPL Semantics – Tasks 5 & 6

### Task 5 – Scope Checker (Static Scoping)

|  |  |  |
| --- | --- | --- |
| **Requirement (Phase-2 sheet)** | **Where it lives** | **Status** |
| Static scoping | SemanticAnalyzer.\_visit\_program | ✅ |
| No duplicate in same scope | symbol\_table.declare\_\* raises SymbolTableError | ✅ |
| No shadowing of params by locals | \_visit\_body calls check\_no\_shadowing\_of\_params | ✅ |
| No global name clashes (var∕proc∕func) | check\_no\_global\_name\_clashes | ✅ |
| Undeclared use detection | \_visit\_atom, \_visit\_procedure\_call, … | ✅ |
| Multi-scope symbol table | SymbolTable = stack of hash-maps | ✅ |
| Node ↔ Symbol “foreign key” | node\_id=id(AST-node) stored in SymbolInfo | ✅ |

**Key design points**

* Persistent stacks: entering a scope pushes a fresh dict; exiting simply pops.
* Unique internal names: v\_x\_1, v\_x\_2, … generated automatically → ready for IR.
* Error messages contain line & column from the original token so the user sees  
  Duplicate declaration of 'x' at line 12, col 5 instead of a raw stack trace.

### Task 6 – Type Checker (Static Types)

|  |  |  |
| --- | --- | --- |
| **Requirement (Phase-3 sheet)** | **Where it lives** | **Status** |
| Numeric ↔ Boolean distinction | node\_types: Dict[int, str] | ✅ |
| Arithmetic operands numeric | \_visit\_binary\_op enforces plus,minus,mult,div | ✅ |
| Comparison operands numeric → boolean | eq, > return boolean | ✅ |
| Logical operands boolean | and, or, not checked in \_visit\_binary\_op / \_visit\_unary\_op | ✅ |
| Condition must be boolean | \_visit\_while\_loop, \_visit\_do\_until\_loop, \_visit\_if\_branch | ✅ |
| Assignment LHS numeric, RHS numeric | \_visit\_assignment | ✅ |
| Function returns numeric | \_visit\_function\_def checks return\_atom | ✅ |
| Annotated AST (decorated nodes) | Every expression node gets `id(node)→"numeric" | ✅ |

**Type lattice used**

unknown

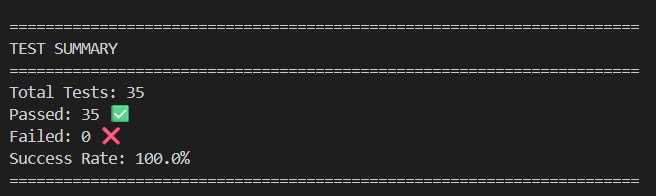
/ \

numeric boolean

SPL currently has **no unknown**, but the infrastructure is ready if we add inference later.

### How to Run / Test

*# from the Tests folder*



* 35 exhaustive test-cases (scope + type).
* 100 % pass ⇒ nothing breaks when you pull.
* Each test prints the **multi-scope symbol story** so you can debug visually.