Scope Design Decision

	Case A	Case B	Case C	Case D	Case E	Case F	Case G
	`num a;` `string a;` `bool a;` `output(a)`	`num a;` `num a;` `output(a)`	`num a;` `a;` `a = 5;` `proc a {` `a = 3;` `a` `}`	`v = 1;` `num v`	`a;` `proc a {` `b` `}` `proc b {` `a` `}`	`a;` `proc a {` `a;` `proc a {` `a` `}` `}`	`a;` `proc a {…}` `proc a {…}`
Is Error		✓		✓			✓
Is Valid	✓		✓		✓	✓	

- Case A:
  Valid - Each declaration of *a* with a different type gets assigned a new variable name, the output will use the last declaration of *a* which is a **bool**. This is such that if the last declaration of a variable is not type-compatible with a following instruction, an error will be thrown in the type checking phase.

- Case B:
  Error - The error occurs on the second declaration of *a* with the same type and same scope as a previous definition.

- Case C:
  Valid - The **num** declaration of *a* and the **proc** definition of *a* will each be assigned different names, and the correct names will be used for the calls and assignments.

- Case D:
  Error – This error will occur on the assignment of *v*. Variable *v* is used chronologically before its declaration, which is not allowed for variables, since they are evaluated sequentially.

- Case E:
  Valid – Each **proc** *a* and *b* is assigned a unique name, and the calls to each other which occur at a higher scope within the respective **prog** code blocks, are allowed because the **proc** declarations occur at a less than (or equal) scope to the calls, and the chronological sequence of **proc** definitions does not apply to declaration checking.

- Case F:
  Valid – Each new **proc** declaration of *a* will be assigned a new unique name, and the calls will be made to the declaration of closest scope (since only one **proc** definition with a particular name is allowed per scope level.)

- Case G:
  Error – This error will occur on the second **proc** definition of *a* because an identifier may only be used once for a certain type on a certain scope level.