

## 4 - bindings and declaration

**identifiers** = given names to program entities (variable, constant, function, type)

↳ most important feature of high level languages

↳ declared once, used  $n$  times

**binding** = finding the corresponding binding definition/declaration (occurrence) for an applied usage (occurrence) of an identifier



☆ the scope of identifiers should be known, to access and not to use same name for other identifiers

**environment** = the set of binding occurrences that are accessible at a point in a program

↳  $\{ \}$  fields  $\rightarrow$  block structure

**blocks**  $\Rightarrow$  define scope and lifetime of the variables

**block structures**

**monolithic** = whole program is a block, all identifiers' scope is global  $\rightarrow$  cobol

**flat block** = global scope + only one single level local scope  $\rightarrow$  fortran, C (partially)

**nested block** = multiple blocks with nested local scopes  $\rightarrow$  Pascal, Java, C

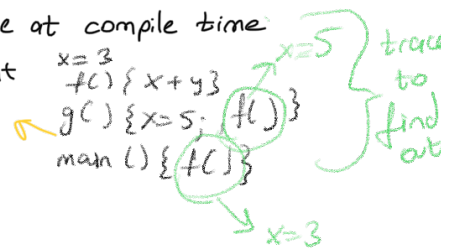
**hiding** = identifiers defined in the inner block hides the outer block identifiers with the same name during their scope (they cannot be accessed within the inner block)

**dynamic/static scope/binding**

**static** = based on lexical structure, binding is done at compile time

**dynamic** = defined during the execution, the environment changes at run time. current stack of activated

block is significant (most recent declaration called from which function)



☆ language processor keeps track of current environment in a symbol/identifier table usually implemented as a hash table. the table maps identifier strings to their type and binding

**definition** = creating a new name for an existing binding

**declaration** = creating a completely new binding.

