Chapter 5 – Names, Bindings, and Scopes

* 5.1 Introduction
  + Imperative programming languages are abstractions of the underlying von Neumann computer architecture
  + Variable can be categorized by collection of properties, or attributes, the most important of which is a type, a fundamental concept in programming languages
  + Functional programming languages allow expressions to be named
* 5.2 Names
  + Design Issues
    - Are names case sensitive?
    - Are the special words of the language reserved words or keywords?
  + Name Forms
    - Name- string of characters used to identify some entity in a program
  + Special Words
    - Used to make programs more readable by naming actions to be performed
    - Reserved word- special word of a programming language that cannot be used as a name
* 5.3 Variables
  + A program variable is an abstraction of a computer memory cell or collection of cells
  + Name- variable names are the most common names in programs
  + Address- machine memory address with which it is associated
    - L-value- address of a variable; named this way because the address is what is required when the name of a variable appears in the left side of an assignment
    - Aliases- when more than one variable name can be used to access the same memory location
  + Type- determines the range of values the variables can store and the set of operations that are defined for values of this type
  + Value- contents of the memory cell or cells associated with the variable
    - R-value- value of a variable; called this because it is what is required when the name of the variable appears in the right side of an assignment statement
* 5.4 The Concept of Binding
  + Binding- association between an attribute and an entity, such as between a variable and its type or value, or between an operation and a symbol
  + Binding time- the time at which a binding takes place
  + Binding of Attributes to Variables
    - Static- if a binding first occurs before run time begins and remains unchanged throughout program execution
    - Dynamic- If the binding first occurs during run time or can change in the course of the program execution
  + Type Bindings
    - Static Type Binding
      * Explicit Declaration- statement in a program that lists variable names and specifies that they are a certain type
      * Implicit declaration- means of associating variables with types through default conventions, rather than declaration statements
      * Type interference- implicit type declarations that use context
    - Dynamic Type Binding- the type of a variable is not specified by a declaration statement, nor can it be determined by the spelling of its name
  + Storage Bindings and Lifetime
    - Allocation- memory cell to which a variable is bound that is taken from a pool of available memory
    - Deallocation- process of placing a memory cell that has been unbound from a variable back into the pool of available memory
    - Lifetime- the time during which a variable is bound to a specific memory location
      * Static variables- those that are bound to memory cells before program execution begins and remain bounds to those same memory cells until program execution terminates
      * Stack-dynamic variables- those whose storage bindings are created when their declaration statements are elaborated, but whose types are statistically bound
        + Elaboration- refers to the storage allocation and binding process indicated by the declaration, which takes place when execution reaches the code to which the declaration is attached
      * Explicit Heap-Dynamic Variables- nameless (abstract) memory cells that are allocated and deallocated by explicit run-time instructions written by the programmer
      * Implicit Heap-Dynamic Variables- bound to heap storage only when they are assigned values
* 5.5 Scope- the range of statements of a variable in which the variable is visible
  + Visible- in a statement, if a variable can be referenced or assigned in that statement
  + Static scope- the method of binding names to nonlocal variables
  + Blocks- a section of code that has its own local variables whose scope is minimized
    - Block-structured language
  + Declaration Order
  + Global Scope
  + Evaluation of Static Scoping- provides a method of nonlocal access that works well in many situations
  + Dynamic scope- based on the calling sequence of subprograms, not on their spatial relationship to each other
  + Evaluation of Dynamic Scoping
* 5.6 Scope and Lifetime
* 5.7 Referencing Environments- the collection of all variables that are visible in the environment
  + Active- if a subprogram’s execution has begun but has not yet terminated
* 5.8 Named Constants- variable that is bound to a value only once
  + Initialization- the binding of a variable to a value at the time it is bound to storage