Chapter 9 – Subprograms

* Introduction
  + Process abstraction and data abstraction are two fundamental abstraction facilities included in a programming language
* Fundamentals of Subprograms
  + General Subprogram Characteristics
    - Each program has a single entry point
      * The calling program unit is suspended during the execution of the called subprogram, which implies that there is only one subprogram in execution at any given time
      * Control always returns to the caller when the subprogram execution terminates
  + Basic Definitions
    - Subprogram definition- describes the interface to and the actions of the subprogram abstraction
    - Subprogram call- explicit request that a specific subprogram be executed
    - Active- subprogram that has begun execution but has not yet completed the execution
    - Subprogram header- first part of the definition, specifies that the following syntactic unit is a subprogram definition of some kind
    - Parameter profile- contains the number, order, and types of its formal parameters
    - Protocol- parameter profile plus its return type, if applicable
    - Prototype- type of parameter that must be checked
  + Parameters
    - Actual- name of the subprogram and a list of parameters to be bound to the formal parameters of the subprogram
    - Positional- first actual parameter is bound to the first formal parameter and so forth
    - Keyword- name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter in a call
  + Procedures and Functions
    - Procedures can produce results in the calling program unit by 2 methods
      * If there are variables that are not formal parameters but are still visible in both the procedure and the calling program unit, the procedure can change them
      * If the procedure has formal parameters that allow the transfer of data to the caller, those parameters can be changed
    - Functions structurally resemble procedures but are semantically modeled on mathematical functions
* Design Issues for Subprograms
  + Overloaded subprogram- same name as another subprogram in the same referencing environment
  + Generic subprogram- computation can be done on data of different types in different calls
  + Closure- nested subprogram and its referencing environment, which together allow the subprogram to be called from anywhere in a program
* Local Referencing Environments
  + Local Variables- variables defined inside subprograms
  + Nested Subprograms
* Parameter-Passing Methods- ways in which parameters are transmitted to and/or from called subprograms
  + Semantics Modes of Parameter Passing
    - In Mode- they can receive data from the corresponding actual parameter
    - Out Mode- they can transmit data to the actual parameter
    - Inout Mode- they can do both
  + Implementation Models of Parameter Passing
    - Pass-by-Value- value of the actual parameter is used to initialize the corresponding formal parameter, which then acts as a local variables in the subprogram, thus implementing in-mode semantics
    - Pass-by-Result- implementation model for out-mode parameters
    - Pass-by-Value-Result- implementation model for inout-mode parameters in which actual values are copid
    - Pass-by-Reference- second implementation model for inout-mode parameters
    - Pass-by-Name- inout-mode parameter transmission method that does not correspond to a single implementation model
  + Implementing Parameter-Passing Methods
  + Parameter-Passing Methods of Some Common Languages
    - Pass-by-assignment- parameter-passing method of Python and Ruby; every variable is a reference to an object
  + Type Checking Parameters
  + Multidimensional Arrays as Parameters
  + Design Considerations
* Parameters that are subprograms
  + Type checking the parameters of the activations of the subprogram that was passed as a parameter
  + Second compilation with parameters that are subprograms appears only with languages that allow nested subprograms
    - Shallow binding- environment of the call statement that enacts the passed subprogram
    - Deep binding- environment of the definition of the passed program
    - Ad hoc binding- environment of the call statement that passed the subprogram as an actual parameter
* Calling Subprograms Indirectly
  + Most often occurs when the specific subprogram to be called is not known until run time
  + Concept of calling subprograms indirectly is not a recently developed concept
  + Multicast delegate- All of the methods stored in a delegate instance are called in the order in which they were placed in the instance
* Design Issues for Functions
  + Are side effects allowed?
  + What types of values can be returned?
  + How many values can be returned?
  + Functional Side Effects
    - Parameters to functions should always be in-mode
  + Types of Returned Values
    - Most languages restrict the types that can be returned by their functions
  + Number of Returned Values
    - Only a single value can be returned from a function, but this may not always be the case
* Overloaded Subprograms- has the same name as another subprogram in the same referencing environment
* Generic Subprograms
  + Polymorphic- takes parameters of different types on different activations
  + Ad hoc polymorphism- overloaded subprograms that need not behave similarly
  + Subtype Polymorphism- variable of type T can access any object of type T or any type derived from T
  + Parametric polymorphism- provided by a subprogram that takes generic parameters that are used in type expressions that describe the types of the parameters of the subprograms
* User-Defined Overloaded Operators
* Closures- a subprogram and the referencing environment where it was defined
  + Unlimited extent- variable whose lifetime is that of the whole program
* Coroutines- special kind of subprogram, caller and called coroutines are not equitable
* Symmetric unit control model- coroutine control mechanisms
* Quasi-concurrency- all the executing programs in such a system appear to run concurrently while sharing the processor