Subprograms

**Introduction**

* Two fundamental **abstraction** facilities
  + Process abstraction
  + Data abstraction

**Fundamentals of Subprograms**

* Each subprogram has a single entry point
* The calling program is suspended during execution of the called subprogram
* Control always returns to the caller when the called subprogram’s execution terminates

**Basic Definitions**

* The **subprogram** **definition** describes the interface to and the actions of the subprogram abstraction
* A **subprogram** **call** is an explicit request that the subprogram be executed
* A **subprogram header** is the first part of the definition, including the name, the kind of subprogram, and the formal parameters
* The **parameter profile** is the number, order and types of its parameters
* The **protocol** is a parameter profile, and if it’s a function, its return type
* A **subprogram declaration** provides the protocol, but not the body of the declaration
* A **formal parameter** is a dummy variable in the subprogram header and used in that program
* An **actual parameter** represents a value or address used in the call statement

**Actual/Formal Parameter Correspondence**

* Positional
  + The binding of actual to formal is by position, the first is bound to the first parameter
  + Safe and effective
* Keyword
  + The name of the formal corresponds to the name of an actual
  + Advantage: Can appear in any order, thus avoiding correspondence errors
  + Disadvantage: User must know the formal parameter names

**Procedures and Functions**

* Subprograms that define parameterized computations:
  + **Procedures** do not return values
  + **Functions** return values

**Local Referencing Environments**

* Local variables can be stack dynamic

**Parameter Passing**

* In mode
  + Passing variables to a called subprogram
  + Can be used to manipulate variables, but doesn’t return anything
  + Actual parameter 🡪 formal parameter
  + **Pass-By-Value**
    - The value of the actual parameter is used to initialize the corresponding formal parameter
* Out mode
  + Pass by result
  + Formal parameter 🡪 actual parameter
  + **Pass-By-Result**
    - No value is passed to the subprogram
    - Formal parameter is a local variable, so its value is transmitted to callers actual parameter when control is returned to the caller
* Inout mode
  + Both, passing the variables to a subprogram, then getting the result from the same subprogram
  + **Pass-By-Value-Result**
    - A combination of the other two
    - Formal parameters have local storage
  + **Pass-By-Reference**
    - Pass an access path
    - Pointers! Yay!!
    - Pass &a, receive \*a, manipulate \*a
* In most languages it happens through the run-time stack
* Pass by reference is the simplest to implement, and most efficient