ICOM 4036  
Programming Languages  

**Subprograms**

- Fundamentals of Subprograms  
- Design Issues for Subprograms  
- Local Referencing Environments  
- Parameter-Passing Methods  
- Parameters That Are Subprograms  
- Overloaded Subprograms  
- Generic Subprograms  
- Design Issues for Functions  
- User-Defined Overloaded Operators  
- Coroutines  

This lecture covers review questions 1-7  

Dr. Amirhossein Chmaei, ECE, UPRM Spring 2010  

---

**Introduction**

- Two fundamental abstraction facilities  
  - Process abstraction  
    - Emphasized from early days  
  - Data abstraction  
    - Emphasized in the 1980s

---

**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**

- A subprogram definition describes the interface to and the actions of the subprogram abstraction  
  - In Python, function definitions are executable; in all other languages, they are non-executable  
- 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 (aka signature) of a subprogram is the number, order, and types of its parameters
Basic Definitions (cont’d)

- The *protocol* is a subprogram’s parameter profile and, if it is a function, its return type.
- Function declarations in C and C++ are often called *prototypes*.
- A *subprogram declaration* provides the protocol, but not the body, of the subprogram.
- A *formal parameter* is a dummy variable listed in the subprogram header and used in the subprogram.
- An *actual parameter* represents a value or address used in the subprogram call statement.

Actual/Formal Parameter Correspondence

- **Positional parameters**
  - The binding of actual parameters to formal parameters is by position: the first actual parameter is bound to the first formal parameter and so forth.
  - Safe and effective.
- **Keyword parameters**
  - The name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter.
  - **Advantage**: Parameters can appear in any order, thereby avoiding parameter correspondence errors.
  - **Disadvantage**: User must know the formal parameter’s names.

Formal Parameter Default Values

- In certain languages (e.g., C++, Python, Ruby, Ada, PHP), formal parameters can have default values (if no actual parameter is passed).
  - In C++, default parameters must appear last because parameters are positionally associated.
- **Ada** Example:

```ada
Function Compute_Pay(income : Float;
                      Exemptions : Integer :=1;
                      TaxRate : Float) return Float;
```

- Some languages (e.g. C, C++, JavaScript, Perl) allow a variable number of parameters.

Procedures and Functions

- There are two categories of subprograms:
  - **Procedures** are collection of statements that define parameterized computations.
  - **Functions** structurally resemble procedures but are semantically modeled on mathematical functions.
    - Can be used to define new user-defined operators.
    - They are expected to produce no side effects.
    - In practice, program functions have side effects.
- C-based languages have only functions.
Design Issues for Subprograms

- Are local variables static or dynamic?
- Can subprogram definitions appear in other subprogram definitions? "nested subprograms"?
- What parameter passing methods are provided?
- Are parameter types checked?
- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?
- Can subprograms be overloaded?
- Can subprogram be generic?

Local Referencing Environments

- Local variables can be stack-dynamic
  - Advantages
    - Support for recursion
    - Storage for locals is shared among some subprograms
  - Disadvantages
    - Allocation/de-allocation, initialization time
    - Indirect addressing
    - Subprograms cannot be history sensitive
- Local variables can be static
  - Advantages and disadvantages are the opposite of those for stack-dynamic local variables

Models of Parameter Passing

Conceptual Models of Transfer

- Physically move a value
- Move an access path