7.1 Fundamentals of Subprograms

- *General characteristics of subprograms*:

1. A subprogram has a single entry point

2. The caller is suspended during execution of the called subprogram

3. Control always returns to the caller when the called subprogram's execution terminates

- *Basic definitions:*

- A *subprogram definition* is a description of the actions of the subprogram abstraction

- A *subprogram call* is an explicit request that the subprogram be executed

- A *subprogram header* is the first line of the definition, including the name, the kind of subprogram, and the formal parameters

- The *parameter profile* of a subprogram is the number, order, and types of its parameters

- The *protocol* of a subprogram is its parameter profile plus, if it is a function, its return type

7.1 Fundamentals of Subprograms
(continued)

- **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:*

1. *Positional*

2. *Keyword*

e.g.
```
SORT(LIST => A, LENGTH => N);
```

***Advantage*: order is irrelevant**
***Disadvantage*: user must know the formal parameter's names**

- *Default Values:*

e.g.
```
procedure SORT(LIST : LIST_TYPE;
               LENGTH : INTEGER := 100);
...
SORT(LIST => A);
```

7.1 Fundamentals of Subprograms (continued)

- Procedures provide user-defined statements
- Functions provide user-defined operators

7.2 Design Issues for Subprograms

1. What parameter passing methods are provided?
2. Are parameter types checked?
3. Are local variables static or dynamic?
4. What is the referencing environment of a passed subprogram?
5. Are parameter types in passed subprograms checked?
6. Can subprogram definitions be nested?
7. Can subprograms be overloaded?
8. Are subprograms allowed to be generic?
9. Is separate or independent compilation supported?

7.3 Local referencing environments

- ***If local variables are stack-dynamic:***

  - ***Advantages:***
    a. **Support for recursion**
    b. **Storage for locals is shared among some subprograms**

  - ***Disadvantages:***
    a. **Allocation/deallocation time**
    b. **Indirect addressing**
    c. **Subprograms cannot be history sensitive**

- ***Static locals are the opposite***

- ***Language Examples:***

  1. **FORTRAN 77 and 90 - most are static, but the implementor can choose either (User can force static with `SAVE`)**

  2. **C - both (variables declared to be `static` are) (default is stack dynamic)**

  3. **Pascal, Java, and Ada - dynamic only**

7.4 Parameter Passing Methods

- **We discuss these at several different levels:**
  - ***Semantic Models:*** **in mode, out mode, inout mode**
  - ***Conceptual Models of Transfer:***
    1. **Physically move a value**
    2. **Move an access path**
- ***Implementation Models:***

1. Pass-by-value (in mode)

- **Either by physical move or access path**
- **Disadvantages of access path method:**
  - **Must write-protect in the called subprogram**
  - **Accesses cost more (indirect addressing)**
- **Disadvantages of physical move:**
  - **Requires more storage (duplicated space)**
  - **Cost of the moves (if the parameter is large)**

7.4 Parameter Passing Methods (continued)

2. Pass-by-result (out mode)

- Local's value is passed back to the caller

- Physical move is usually used
- Disadvantages:
a. If value is passed, time and space
b. In both cases, order dependence may be a problem
e.g.

```
procedure sub1(y: int, z: int);
  ...
sub1(x, x);
```

Value of `x` in the caller depends on order of assignments at the return

3. Pass-by-value-result (inout mode)

- Physical move, both ways

- Also called pass-by-copy

- Disadvantages:
- Those of pass-by-result
- Those of pass-by-value

7.4 Parameter Passing Methods (continued)

4. Pass-by-reference (inout mode)

- Pass an access path
- Also called pass-by-sharing

- *Advantage*: passing process is efficient (no copying and no duplicated storage)

- *Disadvantages*:
  a. Slower accesses
  b. Allows aliasing:
    i. *Actual parameter collisions:*
      e.g.
```
procedure sub1(a: int, b: int);
...
    sub1(x, x);
```

    ii. Array element collisions:
      e.g.
```
sub1(a[i], a[j]);  /* if i = j  */
```
      Also, `sub2(a, a[i]);` (a different one)

    iii. Collision between formals and globals
      - Root cause of all of these is: The called subprogram is provided wider access to nonlocals than is necessary

      - Pass-by-value-result does not allow these aliases (but has other problems!)

8.4 Parameter Passing Methods
(continued)

5. Pass-by-name (multiple mode)

- By textual substitution

- Formals are bound to an access method at the time of the call, but actual binding to a value or address takes place at the time of a reference or assignment

- Purpose: flexibility of late binding

- *Resulting semantics*:

- If actual is a scalar variable, it is pass-by-reference
- If actual is a constant expression, it is pass-by-value
- If actual is an array element, it is like nothing else
e.g.

```
procedure sub1(x: int; y: int);
  begin
 x := 1;
  y := 2;
  x := 2;
  y := 3;
  end;

sub1(i, a[i]);
```

7.4 Parameter Passing Methods (continued)

- If actual is an expression with a reference to a variable that is also accessible in the program, it is also like nothing else

e.g. (assume `k` is a global variable)

```
procedure sub1(x: int; y: int;
                z: int);
  begin
  k := 1;
  y := x;
  k := 5;
  z := x;
  end;

sub1(k+1, j, i);
```

- ***Disadvantages of pass by name:***
  - **Very inefficient references**
  - **Too tricky; hard to read and understand**

- ***Language Examples:***

1. FORTRAN
- **Before 77, pass-by-reference**
- **77 - scalar variables are often passed by value-result**

2. ALGOL 60
- **Pass-by-name is default; pass-by-value is optional**

7.4 Parameter Passing Methods (continued)

3. **ALGOL W**
   - Pass-by-value-result
4. **C**
   - Pass-by-value
5. **Pascal and Modula-2**
   - Default is pass-by-value; pass-by-reference is optional
6. **C++**
   - Like C, but also allows reference type parameters, which provide the efficiency of pass-by-reference with in-mode semantics
7. **Ada**
   - All three semantic modes are available
   - If `out`, it cannot be referenced
   - If `in`, it cannot be assigned
8. **Java**
   - Like C++, except only references

- *Type checking parameters*
(Now considered very important for reliability)

- FORTRAN 77 and original C: none
- Pascal, FORTRAN 90, Java, and Ada: it is always required
- ANSI C and C++: choice is made by the user

7.4 Parameter Passing Methods
(continued)

- *Implementing Parameter Passing*

- *ALGOL 60 and most of its descendants* use the run-time stack

- Value - copy it to the stack; references are indirect to the stack
- Result - same
- Reference - regardless of form, put the address in the stack

- Name - run-time resident code segments or subprograms evaluate the address of the parameter; called for each reference to the formal; these are called *thunks*
- Very expensive, compared to reference or value-result

Ada
- Simple variables are passed by copy (value-result)
- Structured types can be either by copy or reference
- This can be a problem, because
a) Of aliases (reference allows aliases, but value-result does not)
b) Procedure termination by error can produce different actual parameter results
- Programs with such errors are "erroneous"

7.4 Parameter Passing Methods (continued)

- ***Multidimensional Arrays as Parameters***
  - **If a multidimensional array is passed to a subprogram and the subprogram is separately compiled, the compiler needs to know the declared size of that array to build the storage mapping function**
  - **C and C++**
    - **Programmer is required to include the declared sizes of all but the first subscript in the actual parameter**
      - **This disallows writing flexible subprograms**
      - **Solution: pass a pointer to the array and the sizes of the dimensions as other parameters; the user must include the storage mapping function, which is in terms of the size parameters (See example, p. 371)**
  - **Pascal**
    - **Not a problem (declared size is part of the array's type)**
  - **Ada**
    - **Constrained arrays - like Pascal**
    - **Unconstrained arrays - declared size is part of the object declaration (See example p. 371) (Java is similar)**