Introduction to Computers and Programming

Prof. I. K. Lundqvist

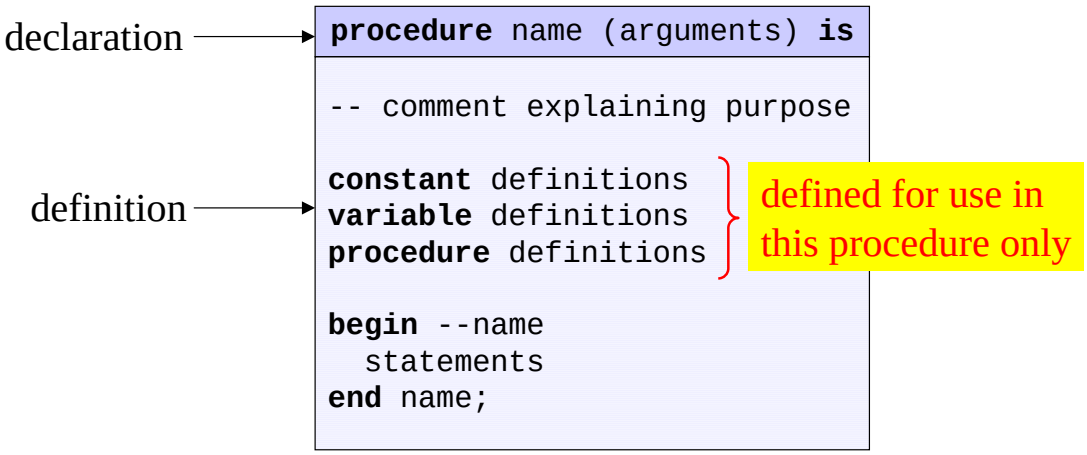
Reading: B pp. 228-234, FK pp. 136-150, 182-193, 276-285

Lecture 9
Sept 19 2003

Structuring Programs

- Mechanisms to control complexity
  - Abstraction
  - Modularization
  - Encapsulation
- Modularity
  - Partition system into modules
    - Reduce complexity, easier development/maintenance
    - Parallel development, divide programming job for teams
  - Ada modules
    - Subprograms: **procedures**, **functions**
    - **Packages**

Procedures



Example

```
procedure display is
    -- display a number
  num : integer;
  begin --show_answer
   num:= 71;
   new_line;
   put("the number of students is :");
   put(num);
   new_line;
  end display;
```

Procedure Call

- Write its name
- Include arguments in brackets

```
begin
   get_two_nums;
   add_two_nums;
   show_answer;
end;
```

- Procedure must be **visible**
  - Declared earlier
  - Included via **`with`**

Procedure Call and Return

- Procedure call
  - Remember where we are in **calling** code
  - Transfer to **called** procedure
    - Set up storage for local variables
    - Associate parameters with values
    - Start execution at first statement of callee
- Procedure finishes executing
  - Wind up **called** procedure
    - Return value through parameter
    - Dispose of storage
  - Pick up where left of in **caller**

Functions

- Effect is to compute a single result
- Returns the result directly
- Function definition: like procedure, except
  - *function* instead of *procedure* as first word
  - Define data type of returned value
  - Include statements to return a value
    `return statement;`
  - Shows which value to return
  - Causes immediate termination of the function
  - The type of the returned value must match the type specified in the function definition
  - There cannot be an execution path through the function that does not include a return statement

Example

```
-----------------------------------------
-- abs: absolute value
-----------------------------------------
function abs (x : in INTEGER) return INTEGER is

begin -- abs
  if x >= 0 then
    return x;
  else
    return -x;
  end if;
end abs;
```

```
y := abs (x);
y := 10 * abs (-4);
y := abs (10 - abs (x));
```

Procedures with Parameters

- Parameters (argument to a procedure)
  - The procedure **declaration** shows the number and type of arguments
    - Formal parameter
  - The procedure **call** supplies specific arguments
    - Actual parameter
- Parameter modes
  - Indicate how data may be communicated between calling and called procedure

Formal Parameters

- Procedure declaration defines **formal** parameters
  - general rules for every call to procedure
    - Mode: **in**, **out**, **in out**
    - data type: integer, character, ...
    - internal name: (for use inside procedure)
  - In brackets after procedure name
  -

```
procedure adjust (
  exam : in INTEGER;     -- exam mark
  mark : in out INTEGER -- overall subject mark
            )
is
  -- local declarations
begin
  -- statements
end adjust;
```

Actual Parameters

- procedure **call** includes **actual parameters**
  - *specific* parameter values for *this* call
  - can differ for each call
- `GET ( val );`

```
get_integer ( exam, 0, 50 );
get_integer ( number, 1, 5 );
get_integer ( number, low, low+4 );
```

```
begin
   get_exam (exam);
   get_lab (labs);
   mark := exam + labs;
   adjust (exam, mark);
   PUT (mark);
   print_grade (mark);
end;
```

Function or Procedure?

```
-----------------------------------------
-- abs: absolute value
-----------------------------------------
procedure abs ( x : in INTEGER; -- argument
                y : out INTEGER -- abs(argument)
              ) is
begin -- abs
  if x >= 0 then
    y := x;
  else
    y := -x;
  end if;
end abs;
```

```
abs (x,y);          -- y := abs(x);

abs (-4, temp);     -- temp:= abs(-4);
y := 10 * temp;     -- y:= 10*abs(-4)
```

Parameter Modes

- Named from perspective of called procedure
  - **`in`** supplied to procedure by its caller
  - **`out`** provided by procedure to its caller
  - **`in out`** supplied to procedure by caller, (possibly) modified, and handed back