Subprograms

- A subprogram allows process abstraction (as opposed to data abstraction).

- Characteristics
  - single entry point
  - caller suspended until control returns from subprogram
  - control returns to caller when subprogram terminates
    - only if subprogram terminates normally
    - exceptions allow for alternate control flow

- Types
  - Functions: return a value
  - Procedures: do not return a value
  - Predicates: are provable (or not)
Parameters

• Formal parameters/parameters
  – in subprogram definition
• Actual parameters/arguments
  – in subprogram call
• Binding of arguments to parameters
  – positional (1\textsuperscript{st} is bound to 1\textsuperscript{st}, 2\textsuperscript{nd} to 2\textsuperscript{nd}, etc)
  – keyword (each parameter has a keyword)
Keyword parameter example

(defun makePoint (&key x y) ...)

(makePoint :x 3 :y 5)
(makePoint :y 5 :x 3)

This example is Lisp. Other languages that permit keyword arguments include Ada and Fortran95.
Default arguments

- C++, Fortran95, Ada and PHP allow default argument values

```c
void example(int x=100, int y)
{
...
}
```

evaluate example(30) ➞ example(100,30)
evaluate example(50,30) ➞ example(50,30)
Variable-length parameter lists

• Some languages allow subprograms to accept a variable number of arguments.
  – Lisp, Scheme, C#, Java

(define f (lambda (x y) ...))
  f takes a list of two args
(define g (lambda lst ...) )
  g takes a list of unspec len
void foo(int x, String... names) {
    System.out.println("x is "+x);
    for (String name:names) {
        System.out.println(name);
    }
}

Vararg must be last parameter.
Parameter passing modes

- **in**: information passes in through parameter
  - pass by value

- **out**: information passes out through parameter
  - pass by result

- **in-out**: information passes both in and out through parameter
  - pass by value-result
  - pass by reference
Ada parameter modes
(Understanding Ada, Bray & Pokrass, pg 71)

There are three parameter modes: in, out and in out. Parameters of mode in are used to pass values (not variables) to a subprogram. Within the body of the subprogram, they are constants and therefore cannot be assigned values. If a parameter declaration does not include an indication of its mode, in is assumed.
Out parameters are used to return values to the caller. Such parameters might be used to return the result of a calculation of a status code. No initial value need be provided by the caller. The subprogram may assign a new value to the parameter but may not refer to it in an expression. That is, the subprogram may not use the value of the parameter, although it can assign a value to it.
The third parameter mode, *in out*, is used to provide an initial value to the called subprogram and return a possibly updated value. A subprogram such as an array sort would use an *in out* parameter for the array to pass in the initial unsorted array and to pass back the same array sorted.
In mode

• This is probably the most familiar and common mode.
• A copy of the argument is passed to the subprogram.
• Argument of subprogram can be any expression producing a value of the correct type:
  – currentPrecip
  – (currentPrecip + previousPrecip) / 2.0
pass by value examples

// Java: assignment has no effect on argument
// double rainfall = 2.0;
// addPrecip(rainfall);
public void addPrecip(double p) {
    p = p * 25.4; // convert from inches to mm
    totalPrecip += p;
}

// C: assignment has no effect on argument
// double rainfall = 2.0;
// addPrecip(rainfall);
void addPrecip(double p) {
    p = p * 25.4; // convert from inches to mm
    totalPrecip += p;
}
out mode

• This is less familiar
• Useful when a subprogram needs to “return” multiple values, but language allows only a single return value from a subprogram.
• Of course, could also return a composite value, such as a tuple, like ML does by default.
Ada example

// two values “returned”
procedure ex(x, y: in INTEGER; s, l: out INTEGER) is begin
    if x<y then
        s = x;
        l = y;
    else
        s = y;
        l = x;
    end if;
end ex;

// Called as follows:
//   a : INTEGER := 5; b : INTEGER := 3;
//   smallest: INTEGER; largest : INTEGER;
//   ex(a, b, smallest, largest);
// Binds smallest to 3, largest to 5
C# example

// two values “returned”
void ex(int x, int y, out int s, out int l) {
    if (x<y) {
        s = x;
        l = y;
    }
    else {
        s = y;
        l = x;
    }
}

// Called as follows:
//   int a=5, b=3, smallest, largest;
//   ex(a, b, out smallest, out largest);
// Binds smallest to 3, largest to 5