Principios de Lenguajes de Programación

Control de subprogramas

Facultad de Informática
Universidad Nacional del Comahue

Primer Cuatrimestre

Control de Secuencias- Resumen

- **Control:** como reglas semánticas (expresadas sintácticamente) para determinar el orden de la ejecución (flujo de control o secuencia de ejecución).
- **Control: Niveles**
  - Al interior de una sentencia (asociatividad y reglas de precedencia)
  - Entre grupo de Sentencias (sentencias de control)
  - Como Abstracción de Grupos de Sentencias (unidades de programas o subprogramas)
- **Control: Tipos**
  - Explícitos
  - Implícitos

Conceptos de Subprogramas: consideraciones

- Cada subprograma tiene un único punto de entrada
- El programa llamador se suspende durante la ejecución del programa llamado
- El control siempre vuelve al llamador, cuando la ejecución del programa invocado termina

Algunas definiciones

- Una ***definición*** *de subprograma* describe la interfase y las acciones del subprograma
- Una ***llamada*** *(o invocación) a un subprograma* es un solicitud explícita de ejecución de un subprograma
- El ***encabezado*** *de un subprograma* es la primera parte de una definición, que incluye nombre, tipo de subprograma y nombre de los parámetros formales.
- La ***signatura*** (o perfil) de un subprograma es el número, orden y tipo de sus parámetros
- El ***protocolo*** de un subprograma especifica cómo debe realizarse la comunicación de parámetros y resultados (tipo y orden de los parámetros y, opcionalmente, valor de retorno)

Algunas definiciones

- La **declaración** de un subprograma muestra el protocolo, pero no el cuerpo.
- Un **parámetro formal** es un identificador listado en el encabezado de un subprograma y utilizado a lo largo del código
- Un **parámetro real (actual)** representa el valor (o expresión) o la dirección utilizada desde la sentencia de llamada al subprograma.

Subprogramas: Categorías

- Existen dos categorías de subprogramas:
  - *Procedimientos*
    - representa un comando que se debe ejecutar
    - colección de sentencias de una computación o cálculo
  - *Funciones*
    - Encarna una expresión que será evaluada
    - estructuralmente similar a los procedimientos
    - Semánticamente modelan funciones matemáticas
    - Normalmente no generan efectos colaterales (en la práctica no se respeta)
  - ***La efectividad en las abstracciones se aumenta mediante la utilización de la parametrización***

Parámetros: correspondencia

- Entre parámetros actuales y formales
- Correspondencia por Posición (o lugar)
  - Se relaciona por posición en la lista de parámetros
  - 1° parámetro formal, 1° parámetro real, etc.
  - Sencillo, seguro y efectivo
- Correspondencia por Identificador o Clave
  - Se especifica el identificador o nombre del parámetro junto al parámetro actual.
  - Se vincula el identificar actual con el identificador formal
  - Aparecen en cualquier orden

Parámetros: correspondencia

```
put(item => 37, base => 8);
put(base => 8, item => 37);
put(37, base => 8);

format_page(columns => 2,
    window_height => 400, window_width => 200,
    header_font => Helvetica, body_font => Times,
    title_font => Times_Bold, header_point_size => 10,
    body_point_size => 11, title_point_size => 13,
    justification => true, hyphenation => false,
    page_num => 3, paragraph_indent => 18,
    background_color => white);
```

Parámetros: valores por defecto

- En C++, Ada y otros, los parámetros formales tienen valor por defecto (en el caso de que la invocación no tenga valor para el parámetro formal)
- En C++, los parámetros por defecto deben aparecer al final de la lista (correspondencia por posición)

Parámetros: valores por defecto

```
Function CalculaPago (
     Ingreso: Float;
     Tasa_Impuestos: Float;
     Excepciones: Integer := 1) return Float;

Function CalculaPago (
     Ingreso: Float;
     Excepciones: Integer := 1;
     Tasa_Impuestos: Float) return Float;

Pay := CalculaPago ( 10000.0,
                     Tasa_Impuestos => 0.15 );
```

Parámetros: valores por defecto

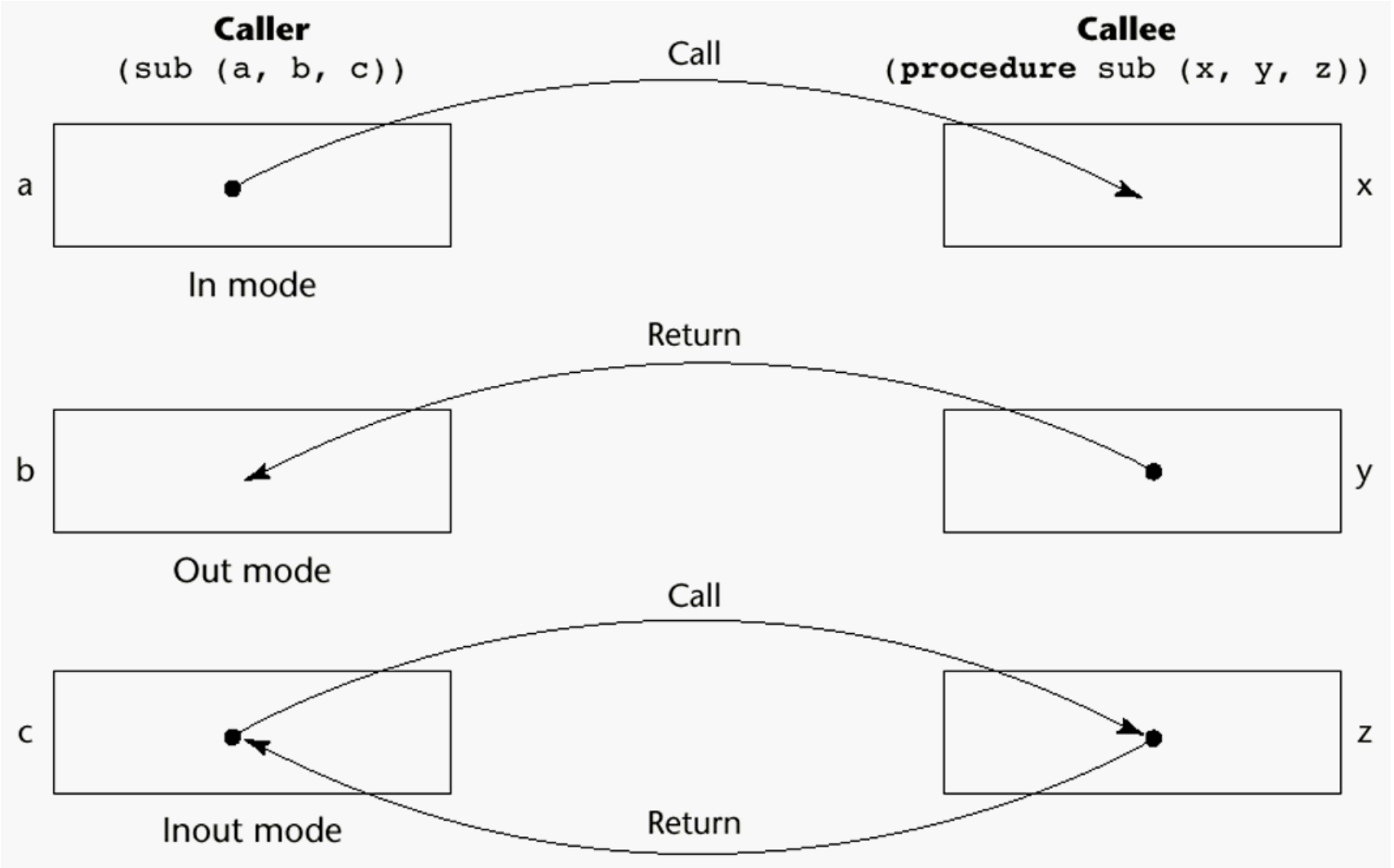
```
float calculapago (float ingreso,
                   float tasa_impuestos,
                   int excepciones = 1 )

Pay = calculapago ( 10000.0, 0.15 );
```

Pasaje de parámetros

- Formas en las cuales los parámetros se transmiten desde y para los subprogramas
  - Pasaje por valor
  - Pasaje por resultado
  - Pasaje por valor-resultado
  - Pasaje por referencia
  - Pasaje por nombre
  - Pasaje por macro

Pasaje de parámetros



Parámetros: Pasaje por Valor

- Modelo In-mode
- El valor del parámetro actual se utiliza para inicializar el valor del parámetro formal correspondiente
- **Normalmente se implementa por copia**
  - Se necesita almacenamiento adicional
- El almacenamiento y las copias pueden ser costosos

Parámetros: Pasaje por Resultado

- Modelo out-mode
- Cuando un parámetro se pasa por resultado **no se transmite un valor** al subprograma **el parámetro formal actúa como una variable local**
  el valor se transmite al llamador cuando el control se devuelve
- Requiere almacenamiento extra y una operación de copia
- Problema potencial:

```
sub(p1, p1);
```

Parámetros: Pasaje por Resultado

```
Void Fixed(out int x,out int Y){
  x = 17;
  y = 35}
…
f.Fixer(out a, out a);
```

- cualquier puede ser el valor final del parámetro
- El orden en el cual los param act son copiados determina el órden de sus valores.

Si x es asignado primero entonces a=35

Si y es asignado primero entonces a=17

Parámetros: Pasaje por Valor-Resultado

- Combinación de pasaje por valor y pasaje por resultado, también llamado Pasaje Por Copia
- Los parámetros formales tienen su propio almacenamiento local
- Desventajas:
  - Las mismas que para Pasaje por Valor
  - Las mismas que para Pasaje por Resultado

Parámetros: Pasaje por Referencia

- Una implementación para el inout-mode
- Se realiza el pasaje de un “camino de acceso” (**puntero**)
- También llamado Pasaje por Compartición
- Eficiente (no hay copia ni almacenamiento duplicado)
- Desventajas:
  - Acceso más lento (comparado con los que tienen almacenamiento propio) de los parámetros formales
  - Origen potencial de problemas de efectos colaterales no deseados
  - Genera “alias”

Parámetros: Pasaje por Nombre

- Toda aparición del parámetro formal en la unidad llamada se reemplaza textualmente por el parámetro actual.
- La ligadura a los objetos se realiza en referencia del espacio del subprograma “llamador”
- Admite mayor flexibilidad para ligaduras tardías
- Hay varios mecanismo para su implementación, aunque en general es un tipo de pasaje de parámetros más costoso que los otros

Parámetros: Pasaje por Macro

- Similar a Pasaje por Nombre
- La ligadura a los objetos se realiza en referencia del espacio del subprograma “actual”, “invocado” o “llamado”
- En caso de referencia a objetos de datos con identificadores que no existen localmente, se comporta como Pasaje por Nombre

Parámetros: Pasaje x Lenguaje

- Fortran
  - Siempre usa semántica in-out (entrada-salida)
  - Antes de Fortran 77: sólo pasaje por referencia
  - Fortran 77 y después: variables escalares normalmente por valor-resultado
- C
  - Pasaje por valor
  - Pasaje por resultado: usando los apuntadores a los objetos de datos como parámetros reales (actuales)
- C++
  - Tipo especial: reference type, para pasaje por referencia
- Java
  - Todos los parámetros se pasan por valor
  - Los objetos se pasan por referencia

Parámetros: Pasaje x Lenguaje

- Ada
  - Tres modelos semánticos: in, out, in out; in (modo default)
  - *out*: asignados pero nunca referenciados
  - *In*: referenciados pero no asignados
  - *In-out*: ambas operaciones
- C#
  - Default: pasaje por valor
  - Pasaje por referencia: especificado con *ref* tanto en los formales como reales
- PHP: similar a C#
- Perl:
  - Todos los parámetros reales se establecen implícitamente en un arreglo predefinido llamado @_