ANÁLISIS COMPARATIVO DE LENGUAJES

Guía de estudio correspondiente a la Teoría sobre: Control de Datos entre Subprogramas

Depto. de Informática

Universidad Nacional de San Luis - Año 2022

CONTROL DE DATOS ENTRE SUBPROGRAMAS

Notas de Clase

Capítulo 9 - Programming Languages – Design and Implementation – Terrence Pratt

CONTROL DE SECUENCIA ENTRE SUBPROGRAMAS

Analizaremos, en mayor detalle, la interacción entre subprogramas y cómo manejan el pasaje de datos entre ellos de manera estructurada y eficiente.

La estructura **call-return** (llamada-retorno simple) de subprogramas es común a la mayoría de los lenguajes de programación. El efecto de esta estructura de control puede ser explicada a partir de la **regla de la copia**:

"el efecto de una sentencia **call** (llamada) a un subprograma, es equivalente al que se obtendría substituyendo esta sentencia por el **cuerpo del subprograma**, con la substitución apropiada de parámetros e identificadores en conflicto, antes de la ejecución.

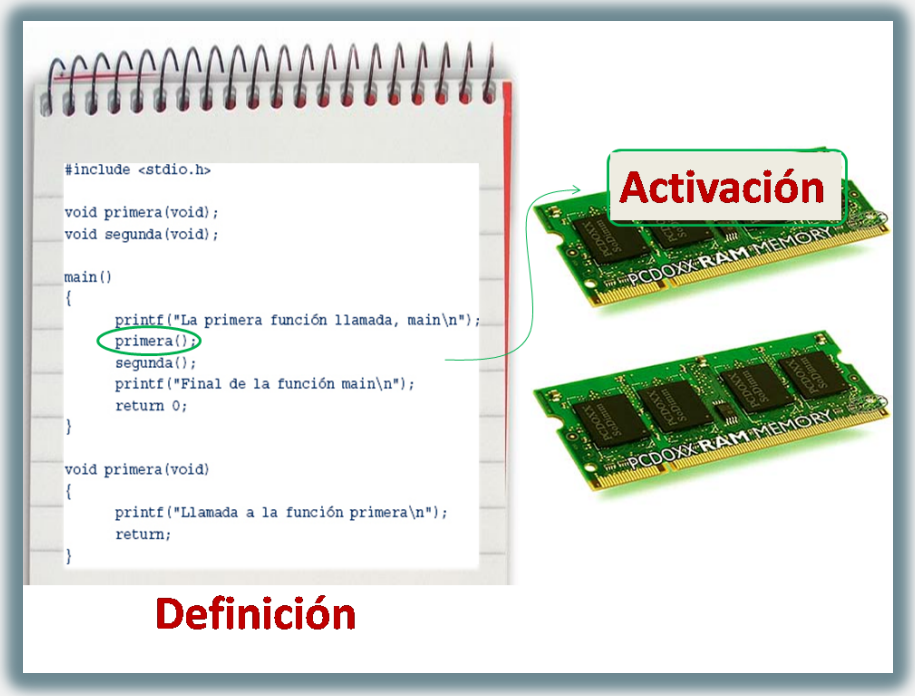
CONTROL DE SECUENCIA ENTRE SUBPROGRAMAS

Existen ciertos supuestos implícitos de la regla de la copia, que al flexibilizarlos dan origen a estructuras de control de subprogramas más generales:

Supuestos de la regla de la copia:

1. Los subprogramas no pueden ser recursivos.
2. Se requiere llamadas explícitas (los manejadores de excepciones no).
3. Los subprogramas deben ejecutarse completamente en cada llamada, o que hay un único punto de entrada (las corutinas no).
4. Transferencia inmediata del control en el punto de llamada (subprogramas planificados la difieren).
5. Secuencia de ejecución única (no ocurre con tareas o subprogramas concurrentes).

DEFINICIÓN Y ACTIVACIÓN DE SUBPROGRAMAS (REPASO)

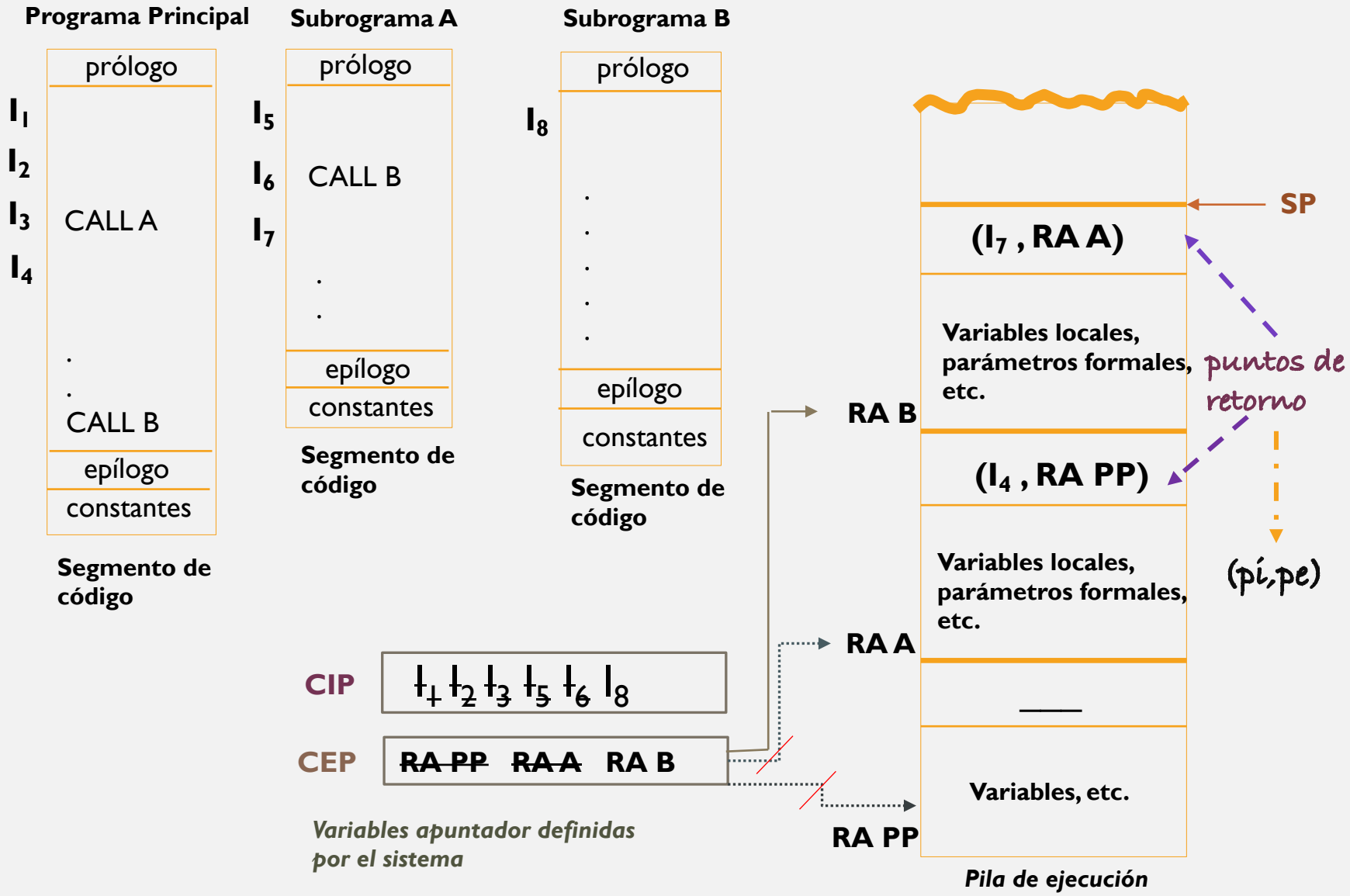


IMPLEMENTACIÓN ESTRUCTURA LLAMADA - RETORNO DE SUBPROGRAMAS

Para mantener el punto en el que el programa se está ejecutando, se requieren dos variables punteros definidas por el sistema:

1. El CIP (Current Intruction Pointer): apunta a la instrucción que está siendo ejecutada (o la que va a ser ejecutada).
2. El CEP (Current Environment Pointer): apunta a la base de registro de activación corriente, es decir, el del subprograma (programa) que se está ejecutando.

IMPLEMENTACIÓN ESTRUCTURA LLAMADA - RETORNO DE SUBPROGRAMAS



SUBPROGRAMAS RECURSIVOS

La recursión es una de las estructuras de control de secuencia más importantes.

Diferencia entre una llamada recursiva y una llamada común: la llamada recursiva crea una segunda activación del subprograma durante el tiempo de vida de la primera (extensible a k activaciones).

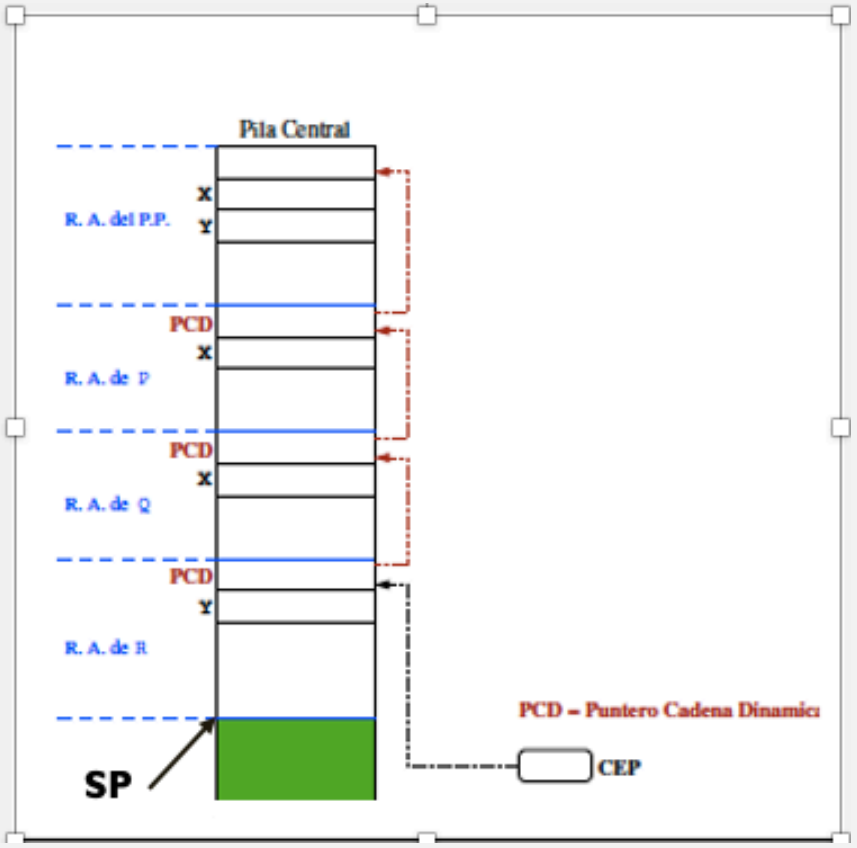
Desde el punto de vista de la implementación:

- Cada registro de activación contiene la pareja de punteros (pi,pe).
- El CEP y los valores de *pe* viejos forman una cadena, que une los R.A. del stack en el orden de su creación (cadena dinámica).
- El uso del stack es solo necesario si tenemos programas recursivos (se pueden plantear enfoques híbridos (tipo PL/I)), pero se utiliza en general para programas no recursivos también, por cuestiones de eficiencia en almacenamiento.

CADENA QUE FORMA LOS VALORES DE CEP VIEJOS

Esta cadena une los registros de activación de subprogramas, ya sean recursivos o no:

```
program Principal
   var X, Y: integer;
   procedure R;
      var Y: real;
      begin
      ...
      X := X + 1; /* Referencia no local a X */
      ...
      end R;
   procedure Q;
      var X: real;
      begin
      ...
      R; /* llamar procedimiento R */
      ...
      end Q;
   procedure P;
      var X: boolean;
      begin
      ...
      Q; /* llamar procedimiento Q */
      ...
      end P;
begin /* comenzar Principal */
...
P; /* llamar procedimiento P */
...
end.
```

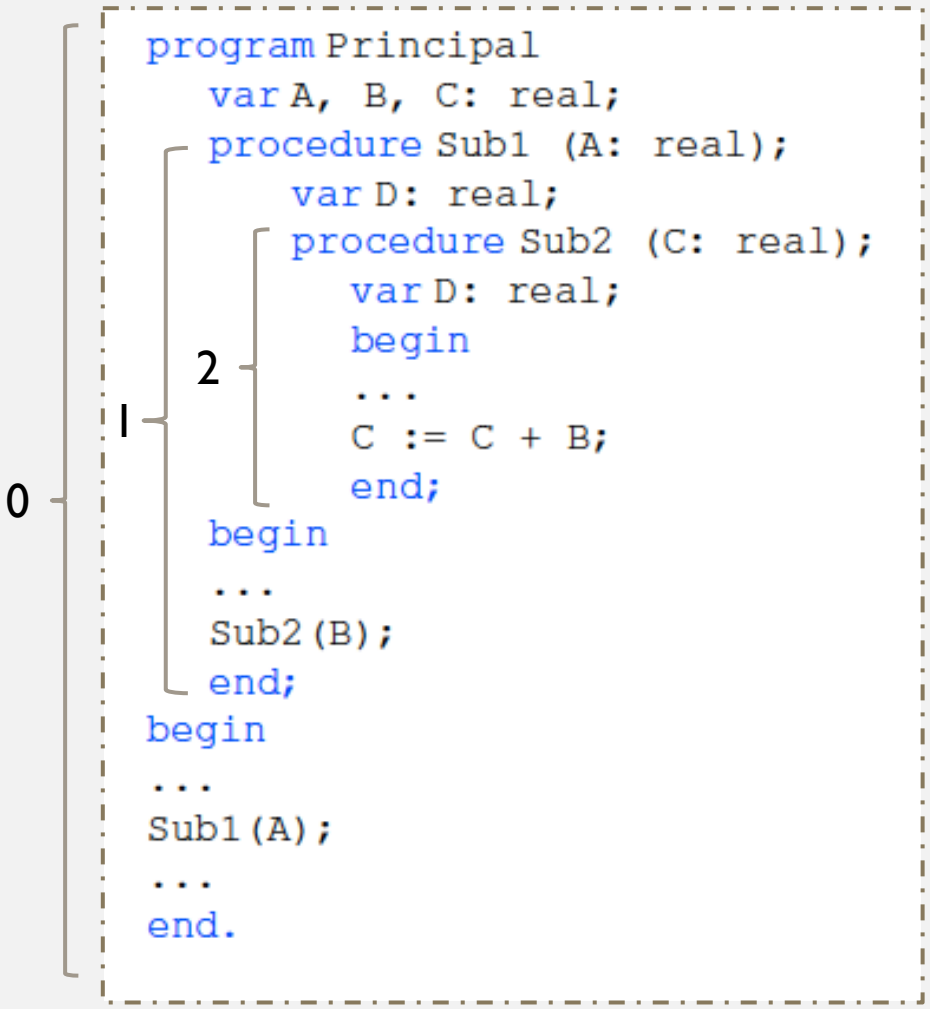


ANIDAMIENTO DE SUBPROGRAMAS

- La idea de anidamientos de subprogramas surgió con Algol 60.
- Este mecanismo provee una forma altamente estructurada para lograr acceso a variables no locales.
- Durante mucho tiempo los lenguajes que permitían anidamiento de subprogramas, fueron aquellos directamente descendientes de Algol 60, los cuales fueron Algol 68, Pascal y Ada.
- Muchos otros lenguajes, incluyendo los descendientes directos de C, no permiten anidamientos de subprogramas.
- Sin embargo en la actualidad algunos lenguajes lo permiten, como Java Script, Ruby, Python y Lua.

EJEMPLO DE ANIDAMIENTOS

Lenguaje Pascal



CONTROL DE DATOS

- Trata con la accesibilidad de los datos en diferentes puntos durante la ejecución del programa.
- El Control de Datos determina cómo acceder y proveer los datos a las operaciones y cómo guardar los resultados para poder ser usados como operandos en otra operación.
- El Control de Datos tiene que ver con el significado de los nombres de los operandos:

```
A = B + 100
```

- `B` puede ser variable local o no local
- `B` puede ser parámetro formal
- `B` puede ser subprograma sin parámetros

ASOCIACIONES Y AMBIENTES DE REFERENCIACIÓN

El Control de Datos trata con la ligadura de identificadores a objetos de datos y subprogramas, a esto llamamos asociación y se representa generalmente como un par $\langle id, OD \rangle$.

El ambiente de referenciación de un subprograma es el conjunto de asociaciones de identificadores que tiene disponibles para usar durante su ejecución. Se establece cuando se crea la activación del subprograma, y permanece sin cambio durante el tiempo de vida de la activación.

COMPONENTES DEL AMBIENTE DE REFERENCIACIÓN DE UN SUBPROGRAMA

- *Ambiente de referenciación local:* parámetros formales, variables locales, subprogramas definidos dentro del subprograma.
- *Ambiente de referenciación no local:* conjunto de asociaciones que pueden usarse en un subprograma, pero no se crean a la entrada del mismo.
- *Ambiente de referenciación global:* es parte del ambiente no local. Se crea al comienzo de la ejecución del programa.
- *Ambiente de referenciación predefinido:* asociaciones definidas directamente en la definición del lenguaje, Ejemplo: `maxint`.

EJEMPLO DE AMBIENTES DE REFERENCIACIÓN

```
program Principal
   var A, B, C: real;
   procedure Sub1 (A: real);
      var D: real;
      procedure Sub2 (C: real);
         var D: real;
         begin
         ...
         C := C + B;
         end;
   begin
   ...
   Sub2(B);
   end;
begin
...
Sub1(A);
...
end.
```

0

1

2

Ambiente de referenciación para Sub2

Local: C, D

No Local: A, Sub2 **en** Sub1

B, Sub1 **en** Ppal

Ambiente de referenciación para Sub1

Local: A, D, Sub2

No Local: B, C, Sub1 **en** Ppal

Ambiente de referenciación para Ppal

Local: A, B, C, Sub1

VISIBILIDAD Y ALIAS (SEUDÓNIMOS)

Visibilidad: una asociación para un identificador es visible dentro de un subprograma si es parte de su ambiente de referenciación.

Alias: Cuando un objeto de datos es visible a través de más de un nombre en el mismo ambiente de referenciación (AR), cada uno de esos nombres se denomina alias del objeto de datos.

EJEMPLO DE ALIAS EN PASCAL

```
program principal
var X:integer;
  procedure P (var Y:integer);
    begin
     Z:=Y;          X,Y son alias en este A.R.
    end
begin
...
P(X);
...
end.
```

EJEMPLO DE ALIAS EN PASCAL

```
program principal
var X:integer;
  procedure P (var Y,Z:integer);
    begin
     Y:=Y+1;
     Z:=X*2;    X,Y,Z son alias en este A.R.
    end
begin
...
P(X,X);
...
end.
```

EJEMPLO DE ALIAS EN C

```
 int *X;
 void P (int *Y) {
    int *Q;
     ...
     Q = Y;   Y,Q,X son alias en este A. R.
     ...
}
void main () {
 ...
 P(X)
 ...
}
```

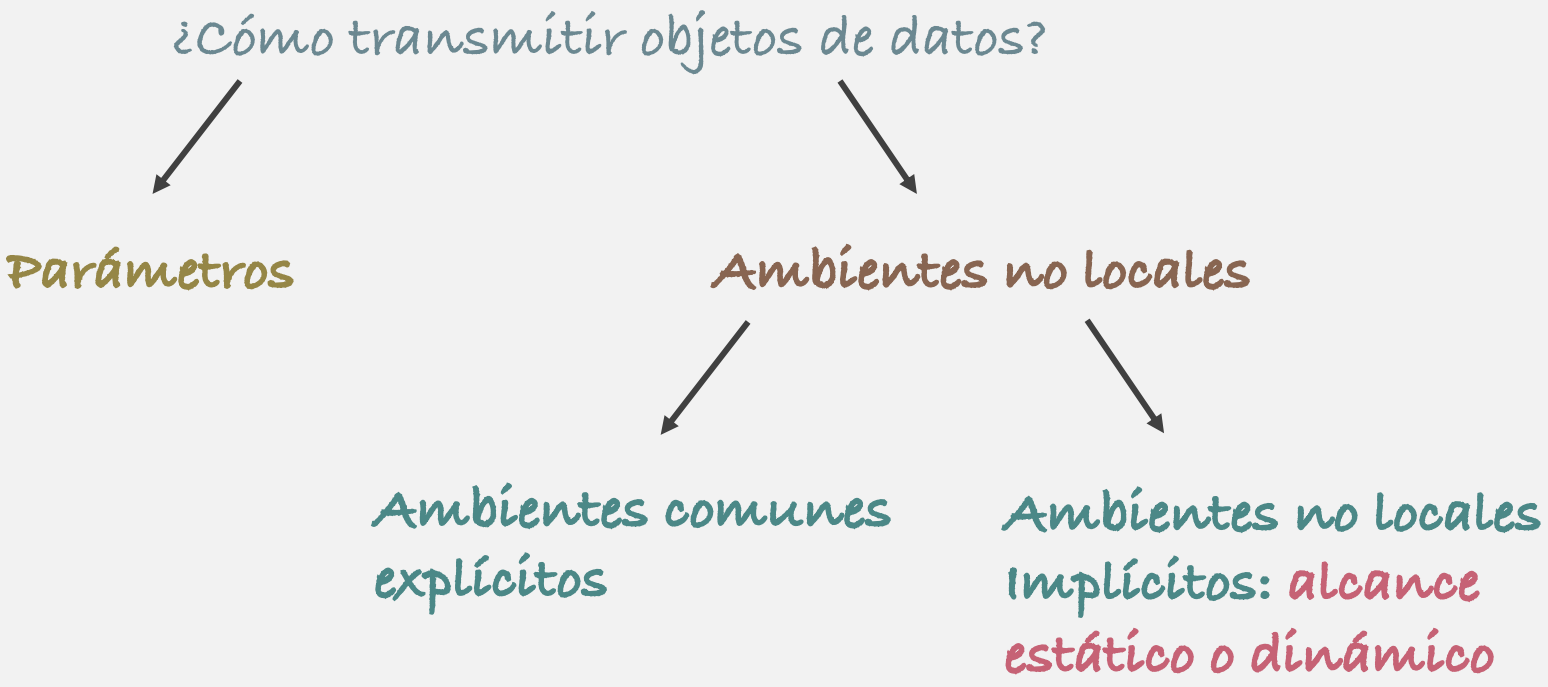
EJEMPLO DE ALIAS EN C++

```
void fun(int &prim, int &seg)

.....
fun(total,total); /* prim, seg en fun son alias */
.....
fun(lista[i],lista[j]);
/* si i, j son iguales, entonces prim y seg
son alias en fun */
.....
```

DATOS COMPARTIDOS EN SUBPROGRAMAS

Un objeto de datos que es local es usado por operaciones sólo dentro del ambiente de referenciación local. Sin embargo, los objetos de datos suelen compartirse entre varios subprogramas.



PARÁMETROS Y TRANSMISIÓN DE PARÁMETROS

Los parámetros y resultados transmitidos de manera explícita constituyen el método alternativo principal para compartir objetos de datos entre subprogramas.

1. **Parámetro Formal:** es una clase particular de objetos de datos local dentro de un subprograma. La definición del subprograma enumera generalmente los nombres y declaraciones para los parámetros formales como parte de la sección de especificación.
2. **Parámetro Actual:** es un objeto de datos que se comparte con el subprograma invocado.

MODELOS SEMÁNTICOS DE PASAJE DE PARÁMETROS

Los parámetros se caracterizan por uno de tres modelos semánticos distintos:

1. Pueden recibir datos desde el correspondiente parámetro actual, llamado **modo in**.
2. Pueden transmitir datos al parámetro actual, **modo out**.
3. Pueden hacer ambas cosas, **modo inout**.

Hay 2 modelos conceptuales de como se realiza la transferencia de datos en la transmisión de parámetros: **o se copia un valor**, o **se transmite un paso de acceso**. Comúnmente el paso de acceso es un puntero o una referencia.

MÉTODOS DE TRANSMISIÓN DE PARÁMETROS

Cuando un subprograma transfiere el control a otro subprograma, debe haber una asociación del parámetro actual del subprograma que llama, con el parámetro formal del subprograma llamado.

Existen varios métodos para transmitir parámetros, de los cuales los más usados son:

- **Por valor:** el valor del parámetro actual se pasa al parámetro formal. Pascal, C, C++, C#, Java.
- **Por dirección o referencia:** el parámetro formal y el parámetro actual referencian a la misma dirección. Pascal, C++, primeras versiones de Fortran, C# (uso de ref).

MÉTODOS DE TRANSMISIÓN DE PARÁMETROS

- **Por resultado:** un parámetro transmitido por resultado se usa sólo para transmitir un resultado al regresar del subprograma invocado. C#
- **Por valor-resultado:** el valor del parámetro actual se copia en el objeto de datos del parámetro formal en el momento de la llamada. Cuando el subprograma termina, el contenido final del objeto de datos del parámetro formal se copia en el parámetro actual. Algol W, Fortran (posterior al 77).
- **Por nombre:** los parámetros son transmitidos sin evaluar. Se reevaluan cada vez que se usan dentro del subprograma. Algol.

EJEMPLO PASAJE DE PARÁMETROS POR DIRECCIÓN Y POR NOMBRE

```
var a: array [1..3] of integer;
 i: integer;
procedure P (X,Y);
begin
 X := X + 1;
 Y := Y + 1;
end;
begin
 ...
 i := 2;
 P(i,a[i]);
end.
```

Por dirección:

i	a[i] -> a[2]
x	y
~~2~~ 3	7

4	7	9

4	8	9

Por nombre:

4	7	9

x	y	i	a[i]
~~2~~ 3	~~9~~ 10	~~2~~ 3	

Cuando se ejecuta: **Y:= Y + 1**, *el valor de* **i** es **3**

4	7	10

¿DUDAS?