

Lenguajes de Programación

Subrutinas y abstracción de control

Ma. Laura Cobo

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación
2018

1

Abstracción

Proceso a través del cual el programador puede asociar un nombre con un fragmento del programa. La idea es pensar la misma en términos de su propósito o funcionalidad en lugar de en términos de su implementación.

- **De control**

su propósito es definir una
operación bien definida

- **De datos**

su propósito es representar
información

(notar que en general para contar con
abstracción de datos hace falta contar con
abstracción de control)

Abstracción

Proceso a través del cual el programador puede asociar un nombre con un fragmento del programa. La idea es pensar la misma en términos de su propósito o funcionalidad en lugar de en términos de su implementación.

- **De control**

su propósito es definir una
operación bien definida



subrutinas

- **De datos**

su propósito es representar
información

(notar que en general para contar con
abstracción de datos hace falta contar con
abstracción de control)

Parámetros

Hay dos maneras de hacerle llegar los datos sobre los cuales computar a una unidad:

- A través del acceso a variables no locales visibles para la unidad
- A través del pasaje de parámetros

Los datos pasados como parámetros son accedidos a través de nombres que son locales a la unidad.

El pasaje de parámetros es más flexible, ya que define computación parametrizada.

El uso de variables no locales puede conducir a programas menos confiables

Parámetros: terminología

- **Argumento:** datos que forman la entrada de una unidad: datos no locales, parámetros y archivos externos (idealmente todos deberían ser parámetros)
- **Resultado:** variables no locales o archivos externos que también pueden modificarse además de los parámetros (se mantienen en el área de almacenamiento temporal , para las funciones no se retornan a través de los parámetros)
- **Parámetro formal:** dato particular dentro del área de referenciamiento de la unidad. Se declaran en el encabezado de la unidad; aclarando en general su tipo
- **Parámetro actual:** variables, constantes, expresiones. Pueden ser locales, no locales o a su vez parámetros.

La elección de que se permite como parámetro actual afecta la implementación del lenguaje de programación

5

Correspondencia entre parámetros formales y actuales

- **Posicional:** la ligadura del parámetro actual con el formal se hace por posición. El primer parámetro actual se liga al primer parámetro formal y así sucesivamente

Es el más habitual (por simple y confiable)

Procedure P(A,B:int; C:real)

Llamada: P(X,Y,Z)

- **Por nombre:** el nombre del parámetro formal al cual se quiere corresponder el parámetro actual aparece junto a éste último

Su ventaja radica en que se independiza del orden, sobre todo cuando hay muchos parámetros, la desventaja radica en que el usuario de la rutina debe conocer los nombre de los parámetros formales.

Se utiliza en general para librerías. Ada admite este tipo de correspondencia.

Llamada: P(B=1,C=0.0,A=100)

6

Correspondencia entre parámetros formales y actuales

La correspondencia por nombre está vinculada con el concepto de **inicialización implícita**.

Así en la invocación no se incluye la lista completa sino solo la de aquellas variables que me interesa que tengan un valor diferente a las establecidas en la definición

Procedure P(A,B:int:=1; C:real:=0)

Llamada: P(A=100)

Algunos lenguajes permiten correspondencia posicional y por nombre, pero no la combinación en una misma llamada

Otros lenguajes permiten posicional y por omisión

Llamada: P(100)

Riesgos si se combina
con sobrecarga

Corresponde posicional, luego se agota por lo que toma los valores de la definición para el resto

7

Tipos de parámetros


Los parámetros pueden ser:

- Datos
- Unidades

Los datos como parámetros:

- Argumentos
- Resultados

Si se trata de datos, el pasaje de parámetros puede ser:

- Referencia
 - Por copia
 - Valor
 - Resultado
 - Valor-Resultado
 - Por nombre
 - Por necesidad
- Evaluación estricta
- Evaluación de orden normal
- Evaluación perezosa
- 

Pasaje de parámetros: por copia

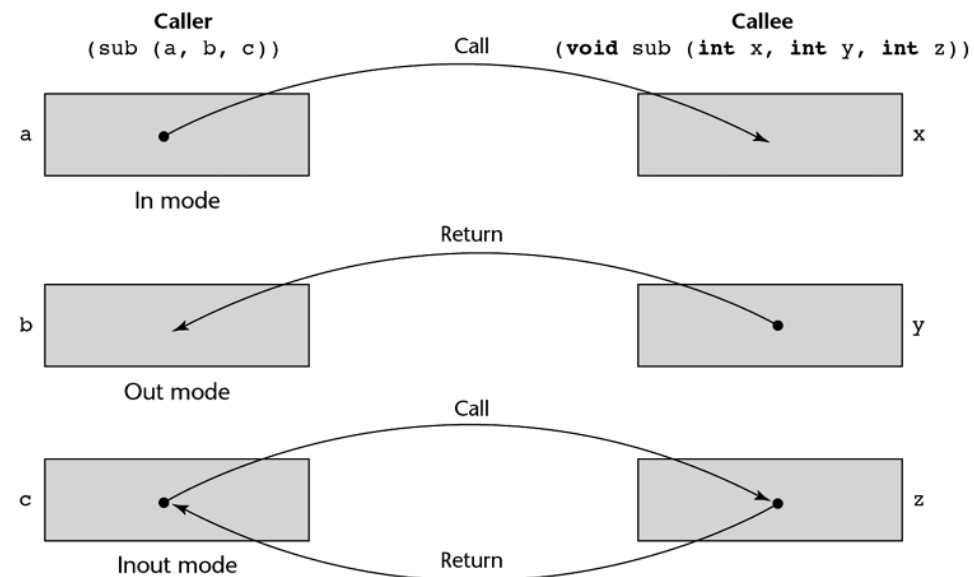
Este tipo de pasaje de parámetros utiliza espacio de memoria adicional para los parámetro formales. Se requiere entonces almacenamiento extra y el uso de operaciones de copia.

- **Por valor:** el parámetro formal se inicializa con el valor del parámetro actual en el momento de la llamada
- **Por resultado:** el parámetro actual recibe el valor del parámetro formal al terminar la ejecución normal de la unidad llamada
- **Por valor-resultado:** combinación de los dos anteriores. El valor del parámetro actual es utilizado para inicializar el parámetro formal, el cual es tratado luego como una variable local. Cuando la unidad termina, se copia el valor del parámetro actual en el formal

Pasaje de parámetros: por copia

Figure 9.1

The three semantics models of parameter-passing when physical moves are used



Implementación: Pasaje de parámetros

Agregemos parámetros a las unidades

- Hay diferentes convenciones para el pasaje de parámetros. La elección está o bien predefinida por el lenguaje o puede ser escogida por el programador de un grupo de opciones. Es importante, conocer cual es la opción elegida, ya que puede afectar el significado del programa.
- Requiere reservar espacio en el registro de activación para los parámetros.
- Afecta las operaciones a realizar en la invocación, prólogo y epílogo de la unidad
- Veremos:
 - a. Pasaje por referencia
 - b. Pasaje por copia (valor , resultado y valor-resultado)
 - c. Pasaje por nombre

Implementación: Pasaje de parámetros

Pasaje por copia

```
Program Principal
Var x:T;
Procedure q(x:T)
  x:= x-1
End q
Procedure p(y:T)
  var z:T;
  y:=3; z:=-3
  call q(y); call q(z)
End p
x:= -11; call p(x);
End Principal.
```

Hay tres formas básicas de pasaje por copia:

- a. Por valor.
- b. Por resultado.
- c. Por valor-resultado.

Implementación: Pasaje de parámetros

Pasaje por copia: valor

```
Program Principal  
Var x:T;  
Procedure q(x:T)  
    x:= x-1  
End q  
Procedure p(y:T)  
    var z:T;  
    y:=3; z:=-3  
    call q(y); call q(z)  
End p  
x:= -11; call p(x);  
End Principal.
```

Con este tipo de pasaje de parámetros, el parámetro formal se piensa como una variable local que se inicializa con el valor del parámetro actual en la invocación de la unidad.

Implementación: Pasaje de parámetros

Pasaje por copia: valor

Program Principal

Var x:T;

Procedure q(x:T)

x := x-1

End q

Procedure p(y:T)

var z:T;

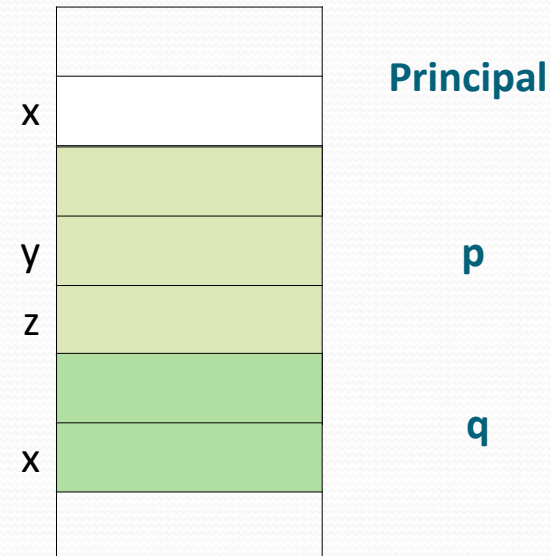
y:=3; z:=-3

call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal.



Implementación: Pasaje de parámetros

Pasaje por copia: valor

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

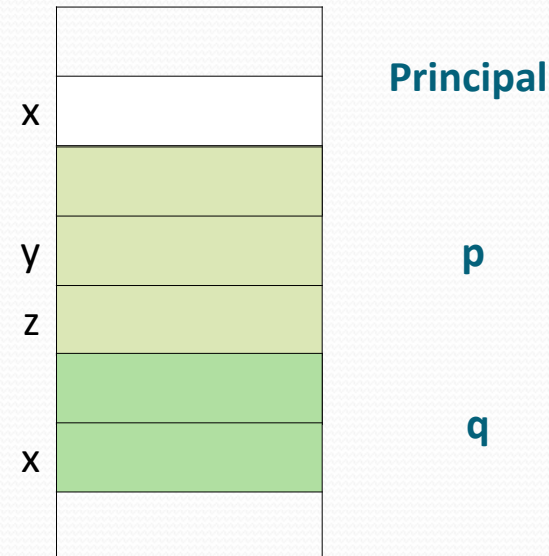
 y:=3; z:=-3

 call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal.



Implementación: Pasaje de parámetros

Pasaje por copia: valor

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; **z:=-3**

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.

Identif	Nivel	Desplaz.	Tipo
x	0	3	local
y	1	3	Valor
z	1	4	local

Segmento código (prólogo – unidad- epílogo)

- 1 SET Actual+3, Pila[Actual+3]-1
- 2 SETR Libre, Actual
- 3 SETR Actual, Pila[Libre+1]
- 4 JUMP Pila[Libre]
- 5 SET Actual+3,3
- 6 SET Actual+4,-3

Implementación: Pasaje de parámetros

Pasaje por copia: valor

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.

Identif	Nivel	Instrucción
q	0	1
p	0	5

Segmento código (invocación)	

7	SET Libre, IP+7
8	SET Libre+1, Actual
9	SET Libre+2, Pila[Actual+2]
10	SET Libre+3, Pila[Actual+3]
11	SETR Actual, Libre
12	SETR Libre, Libre+4
13	JUMP 1

ED
EE
Param.

Implementación: Pasaje de parámetros

Pasaje por copia: resultado

```
Program Principal
Var x:T;
Procedure q(x:T)
  x:= x-1
End q
Procedure p(y:T)
  var z:T;
  y:=3; z:=-3
  call q(y); call q(z)
End p
x:= -11; call p(x);
End Principal.
```

La unidad llamadora es responsable de indicarle a la unidad llamada donde quiere que se guarden los valores de los parámetros, antes de retornar el control.

La unidad llamada lo desconoce, debido a que donde debe guardar los valores se modifica de llamada en llamada.

Implementación: Pasaje de parámetros

Pasaje por copia: resultado

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

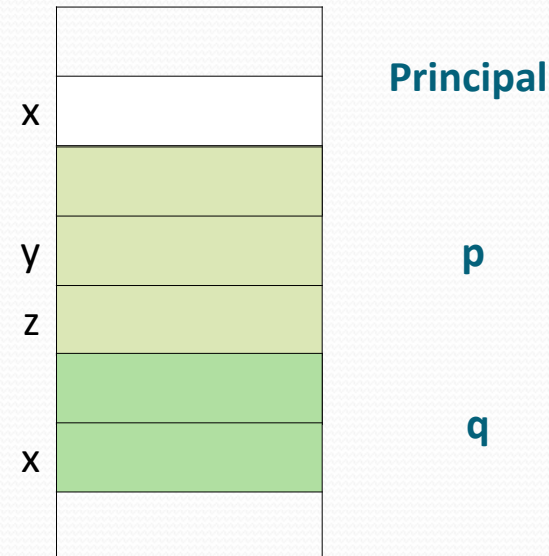
 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

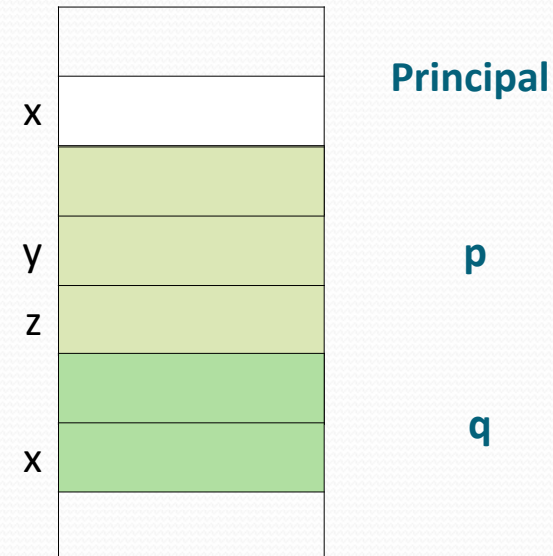
End Principal.



Implementación: Pasaje de parámetros

Pasaje por copia: resultado

```
Program Principal
Var x:T;
Procedure q(x:T)
  x:= x-1
End q
Procedure p(y:T)
  var z:T;
  y:=3; z:=-3
  call q(y); call q(z)
End p
x:= -11; call p(x);
End Principal.
```



Implementación: Pasaje de parámetros

Pasaje por copia: resultado

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.

Segmento código (unidad - epílogo)

- 1 SET Actual+3, Pila[Actual+3]-1
- 2 SET ???, Pila[Actual+3]
- 3 SETR Libre, Actual - 1
- 4 SETR Actual, Pila[Libre+1]
- 5 JUMP Pila[Libre]

Segmento código (invocación)

SET Libre, Actual+3

SET Libre, Libre+1

SET Libre, IP+6

SET Libre+1, Actual

SET Libre+2, Pila[Actual+2]

SETR Actual, Libre

SETR Libre, Libre+4

JUMP 1

Ret

ED

EE

Implementación: Pasaje de parámetros

Pasaje por copia: resultado

Program Principal

Var x:T;

Procedure q(x:T)

x:= x-1

End q

Procedure p(y:T)

var z:T;

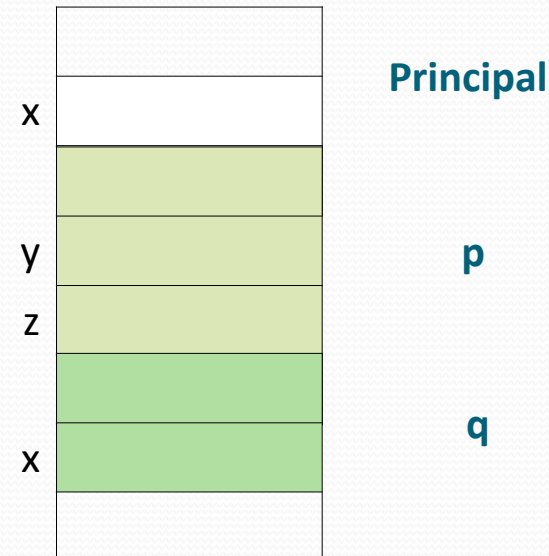
y:=3; z:=-3

call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal.



EPÍLOGO (Copia del resultado)

SET Pila[Actual-1], Pila[Actual+3]

zz

Implementación: Pasaje de parámetros

Pasaje por copia: resultado

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.

Segmento código (cód. unidad - epílogo)

- | | |
|---|------------------------------------|
| 1 | SET Actual+3, Pila[Actual+3]-1 |
| 2 | SET Pila[Actual-1], Pila[Actual+3] |
| 3 | SETR Libre, Actual - 1 |
| 4 | SETR Actual, Pila[Libre+1] |
| 5 | JUMP Pila[Libre] |

Implementación: Pasaje de parámetros

Pasaje por copia: valor-resultado

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

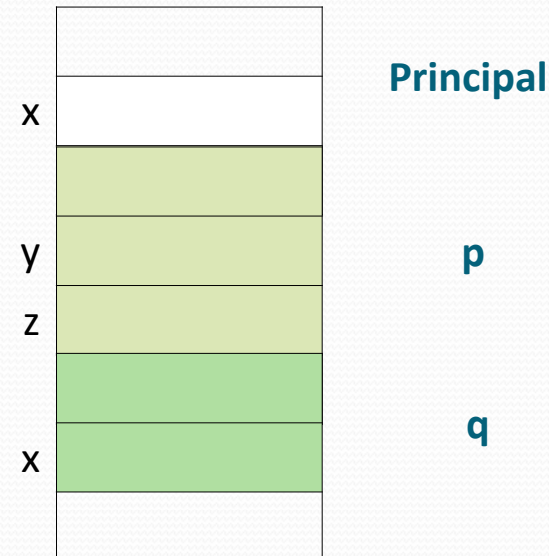
 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.



Implementación: Pasaje de parámetros

Pasaje por copia: valor-resultado

```
Program Principal
Var x:T;
Procedure q(x:T)
  x:= x-1
End q
Procedure p(y:T)
  var z:T;
  y:=3; z:=-3
  call q(y); call q(z)
End p
x:= -11; call p(x);
End Principal.
```

La llamada es igual que en el caso anterior, al igual que el epílogo.

Pero requiere que en el prólogo se realicen las asignaciones de los valores de los parámetros actuales en los formales.

Implementación: Pasaje de parámetros

Pasaje por copia: valor-resultado

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal.

Código Invocación

N+1 SET Libre, Actual+3

N+2 SETR Libre, Libre+1

N+3 SET Libre, IP+6

N+4 SET Libre+1, Actual

N+5 SET Libre+2, Pila[Actual+2]

N+6 SETR Actual, Libre

N+7 SETR Libre, Libre+4

N+8 JUMP 1

Dir de y

Actual. Libre

Retorno

ED

EE

Implementación: Pasaje de parámetros

Pasaje por copia: valor-resultado

Program Principal

Var x:T;

Procedure q(x:T)

x:= x-1

End q

Procedure p(y:T)

var z:T;

y:=3; z:=-3

call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal.

Código Prólogo

N+1

SET Actual+3, Pila[Pila[Actual-1]]

Asig. valor

Código cuerpo unidad

N+2

SET Actual+3, Pila[Actual+3]-1

x:=x-1

Código epílogo

N+3

SET Pila[Actual-1], Pila[Actual+3]

Resultado

N+4

SETR Libre, Actual-1

Retorno de
q

N+5

SETR Actual, Pila[Libre+2]

N+6

JUMP Pila[Libre+1]

Pasaje de parámetros: por referencia

El acceso y modificación del parámetro formal afecta directamente al parámetro actual y por lo tanto al estado del ambiente de referenciamiento de la unidad llamadora.

Su uso puede provocar efectos colaterales y/o aliasing

Implementación: Pasaje de parámetros

Pasaje por referencia

```
Program Principal
Var x:T;
Procedure q(x:T)
    x:= x-1
End q
Procedure p(y:T)
    var z:T;
    y:=3; z:=-3
    call q(y); call q(z)
End p
x:= -11; call p(x);
End Principal.
```

También llamado pasaje “*by sharing*”.

Una referencia al correspondiente parámetro formal en la unidad llamada es tratada como una referencia a la locación cuya dirección ha sido pasada.

Copia la referencia, para ello, antes de la llamada guarda la referencia que va a pasar en la pila.

Implementación: Pasaje de parámetros

Pasaje por referencia

```
Program Principal  
Var x:T;  
Procedure q(x:T)  
    x:= x-1  
End q  
Procedure p(y:T)  
    var z:T;  
    y:=3; z:=-3  
    call q(y); call q(z)  
End p  
x:= -11; call p(x);  
End Principal.
```

Si se tiene pasaje por referencia, y se invoca con una expresión hay dos posibilidades:

- a. El compilador informa un error.
- b. Asigno al parámetro formal, la dirección que tiene asociada la celda que contiene el resultado de la evaluación de la expresión.

Implementación: Pasaje de parámetros

Pasaje por referencia

Program Principal

Var x:T;

Procedure q(x:T)

x:= x-1

End q

Procedure p(y:T)

var z:T;

y:=3; z:=-3

call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal,

Código Invocación

SET Libre, Pila[Actual+3]

SETR Libre, Libre+1

SET Libre, IP+6

SET Libre+1, Actual

SET Libre+2, Pila[Actual+2]

SETR Actual, Libre

SET Libre, Libre+4

JUMP cq

Ref. param. Actual

Actualizar libre

Puntero retorno

Enlace dinámico

Enlace estático

Actualizar actual

Actualizar libre

Ejecutar cod. q

call q(y)

y es un parámetro por referencia,
copia la referencia.

31

Implementación: Pasaje de parámetros

Pasaje por referencia

Program Principal

Var x:T;

Procedure q(x:T)

 x:= x-1

End q

Procedure p(y:T)

 var z:T;

 y:=3; z:=-3

 call q(y); call q(z)

End p

 x:= -11; call p(x);

End Principal,

Código Invocación

SET Libre, Actual+4

Ref. param. actual

SETR Libre, Libre+1

Actualizar libre

SET Libre, IP+6

Puntero retorno

SET Libre+1, Actual

Enlace dinámico

SET Libre+2, Pila[Actual+2]

Enlace estático

SETR Actual, Libre

Actualizar actual

SET Libre, Libre+4

Actualizar libre

JUMP cq

Ejecutar cod. q

call q(z)

Implementación: Pasaje de parámetros

Pasaje por referencia

Program Principal

Var x:T;

Procedure q(x:T)

x:= x-1

End q

Procedure p(y:T)

var z:T;

y:=3; z:=-3

call q(y); call q(z)

End p

x:= -11; call p(x);

End Principal,

Ahora hay que ligar el parámetro formal con el actual, que está en Actual-N

Código Prologo-unidad-epílogo

SET Actual+3, Pila[Actual-1]

SET Pila[Actual+3],
Pila[Pila[Actual+3]]-1

SETR Libre, Actual-1

SETR Actual, Pila[Libre+2]

JUMP Pila[Libre+1]

Le asigno a x la
dirección del
parámetro actual

x:= x-1

Retorno de q

Comienzo de la ejecución de q

Pasaje de parámetros: por nombre

Toda aparición del parámetro formal en la unidad llamada se reemplaza textualmente por el parámetro actual.

El parámetro actual se computa cada vez que el parámetro formal es referenciado.

El cómputo tiene lugar en el ambiente de referenciamiento de la unidad llamadora

Hay varios mecanismos para su implementación, aunque en general es un tipo de pasaje de parámetros más costoso que los otros

Implementación: Pasaje de parámetros

Pasaje por nombre

La técnica básica de implementación consiste en reemplazar cada referencia a un parámetro formal con una llamada a una rutina (thunk) que evalúa una referencia al parámetro actual en el entorno apropiado.

Se crea una rutina para cada parámetro actual.

En cada referencia se llama a la rutina que calcula el valor del parámetro actual, así la sobrecarga en ejecución es importante y hay una pérdida de legibilidad interesante en la traducción.

Se guarda en la pila, la dirección de la primera instrucción de la rutina y ambiente de referenciamiento (actual generalmente)

Pasaje de parámetros: por necesidad

El parámetro actual se computa la primera vez que el parámetro formal es referenciado en una expresión

El cómputo se realiza en el ambiente de referenciamiento de la unidad llamadora

Parámetros: Chequeo de tipos

La consistencia entre parámetros formales y actuales puede resolverse en los lenguajes de programación:

- Sin chequeo
- Chequeo coherente: aplicado a expresiones y asignaciones

Unidades como parámetros

Es conveniente, a veces, pasar unidades como parámetros. Surgen en estos casos algunos aspectos de diseño que deben considerarse:

- Decidir si se controlara el tipo de los parámetros
- Decidir cual será el ambiente de referenciamiento para el subprograma pasado como parámetro

Algunos lenguajes controlan el tipo de los parámetros, por ejemplo Fortran.

En C y C++ no se pueden pasar unidades como parámetro, pero si punteros a unidades. En este último caso se hace chequeo de tipos

Ada no permite pasar unidades como parámetros

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

- Los lenguajes que soportan variables de tipo rutina se dice que tratan a las unidades como objetos de primera clase.
- Si permiten que las mismas sean pasadas como parámetros, esos parámetros reciben el nombre de parámetros procedurales.
- Ésta practica resulta útil en algunas situaciones prácticas, pero involucra la resolución de aspectos que hasta ahora eran inexistentes.

Hasta ahora el llamado a una unidad involucraba un conjunto de instrucciones. Era necesario:

- Reservar el espacio para el registro de activación y setear el enlace estático.
- Ahora es imposible hacerlo, ya que no conocemos en la traducción quién es la rutina que estamos recibiendo como parámetro. Esta información recién se conoce en ejecución.

38

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

Para resolver los problemas que aparecen necesitamos pasarle a la unidad llamada:

- el tamaño del registro de activación y
- el correspondiente enlace estático.

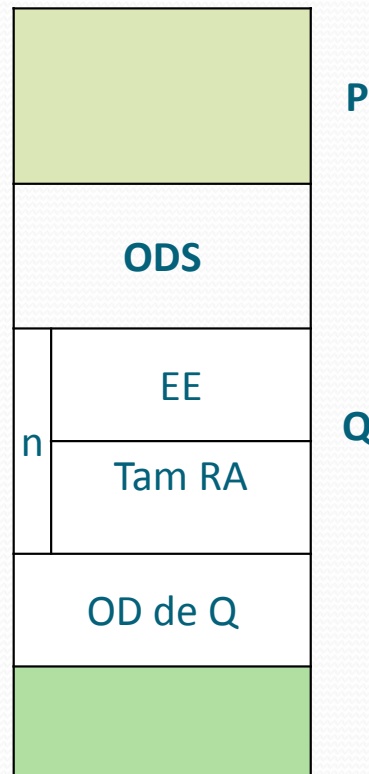
Se puede determinar el enlace a pasar por las reglas de alcance y las reglas que haya decidido utilizar el lenguaje para resolver los accesos no locales en estos casos.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

```

Proc P
  var u,v
  Proc R
    var y
    u
  end R
  Proc Q (proc X)
    var u,v,y
    call X
  end Q
  call Q(R)
End P
    
```



Necesito guardar toda la información sobre la unidad al momento de hacer la llamada, ya que al hacer la traducción del código de Q no se quién es X en ejecución

Parte traducción call Q(R)

ASIG Actual, Pila[Libre,dProcX]
ASIG Tamaño RA R, Pila[Libre+1,dProcX]

.....

Unidades como parámetros

Ambiente de referenciamiento no local:

- **Superficial** (*shallow binding*): el ambiente de referenciamiento de la instrucción de llamado (al parámetro) lo determina.
- **Profundo** (*deep binding*): el ambiente de la definición de la unidad llamada.
- **Ad-hoc**: el ambiente de la sentencia de llamado que pasa la unidad

Veamos un ejemplo de cómo se determina:

Unidades como parámetros

```
Function sub1()  
{ var x;  
  function sub2()  
  {imprimir(x);}  
  function sub3()  
  {  
    var x;  
    x=3  
    sub4(sub2);}  
  function sub4(subx)  
  {  
    var x;  
    x=4;  
    subx();}  
  
  x=1;  
  sub3();  
}
```

La ejecución de sub2 dede sub4:

Superficial:

el ambiente de referenciamiento es el de sub4, por lo tanto se imprime el valor 4.

- Profundo:

el ambiente de referenciamiento es el de sub1, por lo tanto está ligado a la variable local imprimiendo 1

- Ad-hoc:

el ambiente corresponde al de sub3 y por lo tanto sub2 imprime el valor 3

Unidades como parámetros

```
proc s
```

```
  var x:integer;
```

profunda

```
  proc s1
```

```
    imprimir(x)
```

```
  end s1
```

```
  proc s2(proc f)
```

```
    var x: integer;
```

superficial

```
    x=1; f;
```

```
  end s2
```

```
  proc s3
```

```
    var x:integer;
```

Ad-hoc

```
    x=2; s2(s1)
```

```
  end s3
```

```
  x=3; s3;
```

```
end s
```

43

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

Para que una unidad **U** pueda pasar una unidad **A** como parámetro a una unidad **B**, la unidad **U** debe:

- a. Tener la unidad **A** dentro de su alcance, esto es **A** debe ser visible no localmente o ser local (anidada en forma inmediata)
- b. **A** puede ser un parámetro formal de **U**.

Caso a.

El enlace estático a ser pasado es un puntero al registro de activación que es a **D** pasos a lo largo de la cadena estática originada en la unidad llamadora.

Dónde **D** es la distancia entre el punto de llamada en la unidad dónde el parámetro es pasado y su declaración.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

Para que una unidad **U** pueda pasar una unidad **A** como parámetro a una unidad **B**. Se de una de las siguientes alternativas:

- a. **U** debe tener a la unidad **A** dentro de su alcance, esto es **A** debe ser visible no localmente o ser local (anidada en forma inmediata)
- b. **A** puede ser un parámetro formal de **U**.

Caso b.

El enlace estático a ser pasado es el mismo que fue oportunamente pasado al llamador.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

Con respecto a la invocación en sí, la única diferencia entre una invocación a una unidad tradicional y una realizada a un parámetro, es que la información con respecto al tamaño del registro de activación y el enlace estático, se copia desde el área de los parámetros.

El impacto de este agregado a un lenguaje de programación no es menor. Es necesario extender los procedimientos de invocación para poder lidiar con la semántica adicional y la complejidad. Las invocaciones deben tratar con diferentes tipos de objetos. Por lo que todas las partes involucradas sufren un incremento considerable en la complejidad.

Como ventaja se tiene un lenguaje más uniforme, ya que las unidades son tratadas como cualquier otro objeto del lenguaje.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

```
Proc T
  Var v
  Proc Q (proc X)
    var u
    call X
  End q
  Proc P
    var u
    Proc R
      call Q(R)
    end R
  end P
End T
```

En este contexto la traducción es correcta, ya que cuando se realiza la llamada **call Q(R)**, **P** está activo y por lo tanto en la tabla de símbolos tengo a **R**.

En el momento que el compilador traduce, mira en la tabla de símbolos y lo encuentra y permite que realice la llamada.

Liga el EE del **proc X** a **P** y es lógico porque **R** está en el ambiente de **P**.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

```
Proc T
  Var v
  Proc Q (proc X)
    var u
    Proc R
      u,v
    end R
  end Q
  Proc P
    var u
    call Q(R)
  end P
End T
```

La llamada no es correcta, ya que cuando el compilador quiere traducir **R**, ya no está en la tabla de símbolos porque **Q** ya terminó de compilarse y la información sobre **R** ya no está en la tabla de símbolos.

Implementación: Pasaje de parámetros

Procedimientos y funciones como parámetros

```
Proc Q (proc X)
  var u,v,y
  Proc T
    y
  end T
  call X
  call Q (T)
end Q
```

Cuando resuelvo el EE de T, el único RA activo es el primer registro de activación de Q, es por eso que cuando se hace referencia a y en T, hago referencia al que está en el primer RA de Q.