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)

2

Control de secuencia

Las estructuras de control de secuencia puede categorizarse en cuatro grupos:

- **Expresiones**: Las expresiones son la forma básica de construir bloques y expresan como los datos son manipulados y modificados por el programa
- **Sentencias o instrucciones**: determinan como el control fluye de un segmento de programa a otro
- **Programación declarativa**: modelo de ejecución que no depende de sentencias, pero establece como la ejecución tiene lugar (ejemplo **Prolog**)
- **Unidades-Subprogramas**: indican una manera de pasar transferir el control de un segmento de programa a otro.

Esta división no es precisa ya que hay lenguajes que tiene mas de una característica, por ejemplo LISP (computa con expresiones pero usa mecanismos de control de sentencias)

3

Control de secuencia

Las estructuras de control de secuencia pueden ser:

- **Implícitas**: son las definidas por el lenguaje para tener efecto a menos que sean modificadas por el programador a través de una estructura de control explícita
- **Explícitas**: son aquellas definidas por el programador para modificar el control implícito. Por ejemplo el uso de los paréntesis en las expresiones o el uso de la instrucción **goto** a un etiqueta en un programa que solo tiene secuencia

En clases previas ya vimos como se resuelve el control de secuencia a través de expresiones e instrucciones. Veamos las dos alternativas que nos faltan.

4

Control de secuencia: Programación declarativa

Pattern Matching: una operación tiene éxito si ajusta y asigna un conjunto de variables a un patrón predefinido

Es una operación crucial para lenguajes como:

Perl, Snobol, ML, Miranda, Prolog

El pattern maching produce dos efectos:

- Por un lado, como estructura de control, permite decidir que parte del código se va a ejecutar
- Permite establecer ligaduras entre el patrón de la definición y el de la instancia (instancia las variables)

5

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)

6

Control de secuencia: Unidades o subrutinas

Los lenguajes de programación imperativos y orientados a objetos brindan mecanismos para dividir el programa en **unidades** cada una de las cuales tiene cierta coherencia y lógica.

Su incorporación fomenta:

- **Eficiencia**: permiten factorizar el código (en los primeros lenguajes solo se perseguía el incremento de eficiencia principalmente de almacenamiento)
- **Legibilidad**: metodología de diseño top-down (mecanismos más abstractos)
- **Verificación**: una unidad puede pensarse como una mapeo entre dominios de valores. Se intenta que los chequeos sean lo más seguros posibles, así evolucionaron los métodos de pasaje de parámetros.

7

Unidades o subrutinas

Las unidades pueden ser:

- Anónimas (bloques)
- Con nombre asociado (subprogramas)

Un unidad con nombre permite extender el lenguaje con una nueva primitiva.

Para que la abstracción sea poderosa , se debería poder usar conociendo sólo la interfaz . La implementación debería ser transparente

Recordar que la unidad puede ser una abstracción a nivel expresión o una abstracción a nivel instrucción

8

Unidades o subrutinas

El los lenguajes orientados a objetos cada unidad con nombre brinda un **servicio** que puede ser provisto por cualquiera de las instancias de la clase en la cuál se define la unidad.

En este tipo de lenguajes la clase es el mecanismo de abstracción

Un servicio puede acceder y hasta modificar el estado interno del objeto que recibe el mensaje que provoca la ejecución de ese servicio

Aspectos de diseño a tener en cuenta por el LP:

- Entidades
- Ligaduras
- Reglas de alcance
- Sistemas de tipos
- Soporte para definir unidades

9

Unidades o subrutinas

Aspectos de diseño específicos:

- Estructura estática
- Recursividad
- Control
- Métodos de pasaje de parámetros
- Correspondencia entre parámetros formales y actuales
- Sobrecarga y polimorfismo
- Tipos de los parámetros: datos y unidades
- Ambiente de referenciamiento de las unidades que recibe como parámetro (chequeo estático o dinámico de unidades)

10

Unidades o subrutinas

Atributos de la unidad:

- **Nombre**: algunos lenguajes permiten definir unidades anónimas. Las unidades anónimas se ejecutan implícitamente y cuando se alcanzan se crea un nuevo ambiente de referenciamiento.
- **Lista de parámetros**: permiten la comunicación de la unidad con el resto del programa. Si se admiten parámetros de diferentes tipos la unidad es polimórfica.
- **Ambiente de referenciamiento**: se establece a que otras unidades puede referenciar
- **Bloque ejecutable**: si la unidad tiene la capacidad de llamarse a si misma soporta recursividad
- **Alcance**: segmento de código dentro del cual la unidad puede invocarse (si incluye a la misma unidad acepta recursividad)

11

Unidades

Aspectos de implementación:

- Chequeo de tipos entre parámetros
- Unidades de compilación
- Creación del ambiente de referenciamiento local

Estructura estática:

La unidad tiene un **encabezamiento** y un **cuerpo**

El **encabezamiento** está formado por el nombre de la unidad, la lista de parámetros y el tipo del resultado

El **cuerpo** está formado por las declaraciones locales y la sección ejecutable

12

Unidades

Estructura estática:

En el **encabezamiento** se puede definir el **perfil de los parámetros**, el **protocolo** y la **signatura**

El **perfil de los parámetros** de una unidad es el número, orden y tipo de los parámetros formales

El **protocolo** de una unidad es el perfil de los parámetros más el tipo retornado por la unidad, si es que tiene

La **signatura** de una unidad está dada por el nombre de la unidad y su protocolo y define la interfaz de la unidad

13

Unidades

La unidad estáticamente posee:

- Declaración
- Definición
- Invocación

La invocación tiene una vinculación bidireccional con la activación de la unidad y con su suspensión

La unidad dinámicamente puede estar:

- Pasiva
- Activa – Suspendida

Su estado esta determinado por las estructuras de control

14

Unidades

La comunicación entre unidades puede darse a través de:

- Ambiente global
- Ambiente implícito
- Parámetros y resultado

La comunicación a través del ambiente global puede generar efectos colaterales

La comunicación a través del ambiente implícito puede generar comportamiento sensible a la historia

Recordemos que hay efecto colateral si la ejecución de una unidad modifica el ambiente de referenciamiento de la unidad llamadora

15

Mecanismos para definir unidades

- **Bloques**: unidades anónimas
- **Rutinas:**
  - **Abstracciones procedurales**: abstracción a nivel instrucción

    La manera de invocar a este tipo de rutinas es a través de una llamada en la cual aparece el nombre de la rutina

    call <nombre proced>

  - **Abstracciones funcionales**: abstracción a nivel expresión

    La manera de invocar a este tipo de rutinas es en el contexto de una expresión

16

Mecanismos para definir unidades

Una **abstracción procedural** es una unidad que al ser invocada evalúa una instrucción y en general provoca un cambio en el estado de computación, esto es, modifica los valores del ambiente de referenciamiento de la llamada provocando un efecto colateral.

- El usuario de una procedimiento observa únicamente las transformaciones en estado de la memoria pero no los pasos que dieron lugar a esta transformación

Una **abstracción funcional** es una unidad que evalúa una expresión y al ser invocada produce un resultado.

- Idealmente no debería producir un efecto colateral
- Es una manera de hacer abstracción a nivel expresión

17

Mecanismos para definir unidades: funciones

En lenguajes imperativos

```
Function Pot(n,m:int):int
   if n=1 then Pot:=1
          else Pot:= Pot(n,m-1)*n
End;
```

En lenguajes funcionales

```
fun Pot(n,m:int):int
   if n=1 then 1
          else Pot(n,m-1)*n
End;
```

En algunos lenguajes se hace el **return** del valor (El valor computado no se asigna a una variable)

Se utiliza el nombre la función como área de almacenamiento.

Así el nombre de la función tiene dos semánticas dependiendo de dónde aparezca:

1. Área de almacenamiento auxiliar (a la izquierda)
2. Invocación recursiva (a la derecha)

En los lenguajes funcionales se evita la doble semántica sobre el nombre de la función.

18

Representación en memoria

El espacio que se aloca para una subrutina en la pila se conoce como *registro de activación* o *stack frame*.

El registro contiene los argumentos, valores de retorno (en caso que los tenga), variables locales y temporales.

El registro se elimina de la pila cuando la subrutina alcanza su *epílogo*.

Hay dos punteros que mantienen el estado de la pila: **actual** (*frame pointer*) y **libre** (*stack pointer*)

Los objetos del registro de activación se acceden por desplazamiento desde la posición almacenada en **actual**.

19

Representación en memoria

El espacio que se aloca para una subrutina en la pila se conoce como *registro de activación* o *stack frame*.

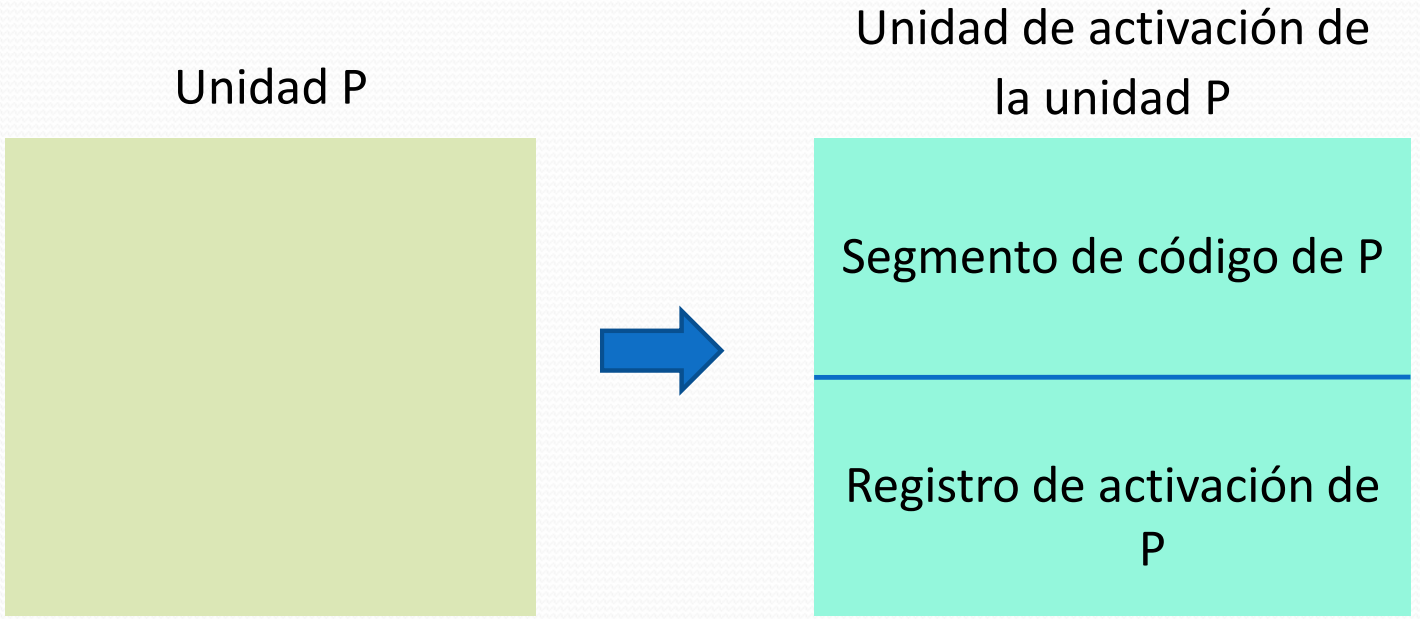
El registro de activación mantiene además información de sistema para garantizar el funcionamiento: puntero de retorno, enlace dinámico ….

En aquellos lenguajes que admiten anidamiento, es necesario conservar la cadena estática a fin de resolver los accesos no locales (enlace estático)

20

Representación en memoria

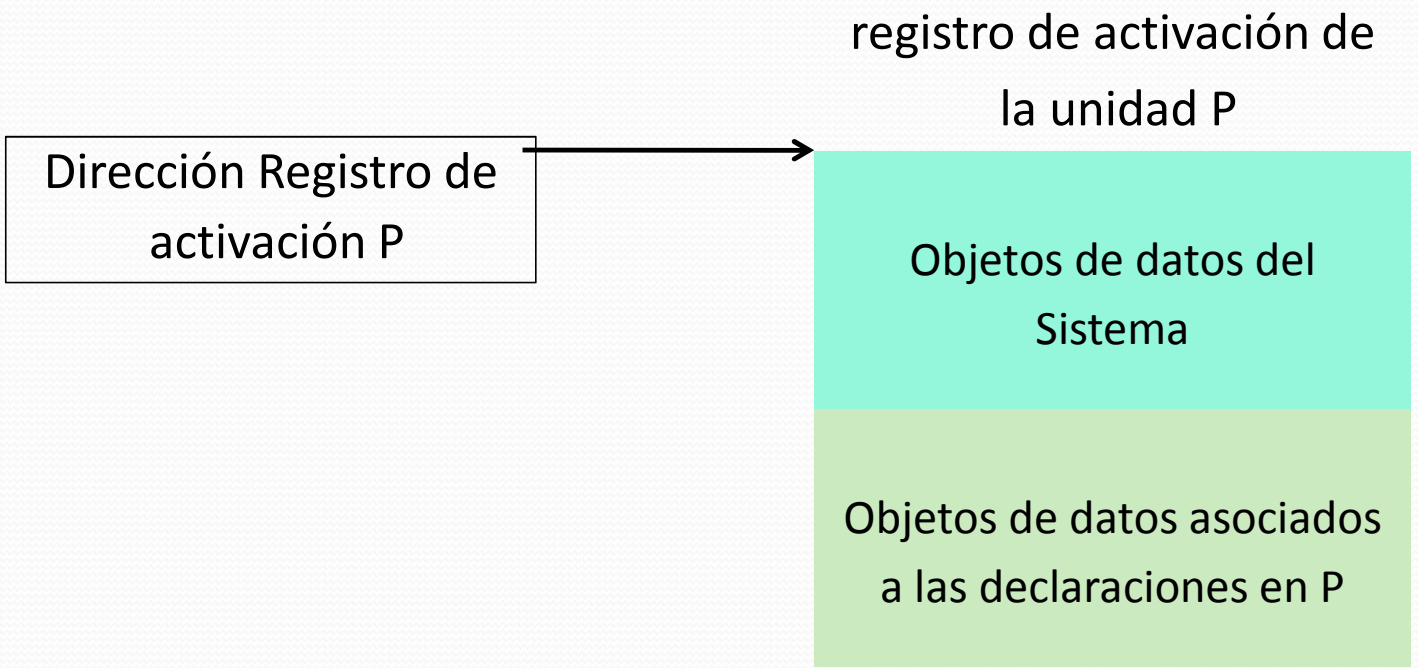
Si nuestro lenguaje considera unidades, nuestro procesador abstracto debe considerar la *unidad de activación de P*. La unidad de activación contiene toda la información relevante en ejecución de la unidad.



21

Representación en memoria

El *registro de activación de P* contiene la información sobre los objetos de datos, del sistema y declarados en la unidad

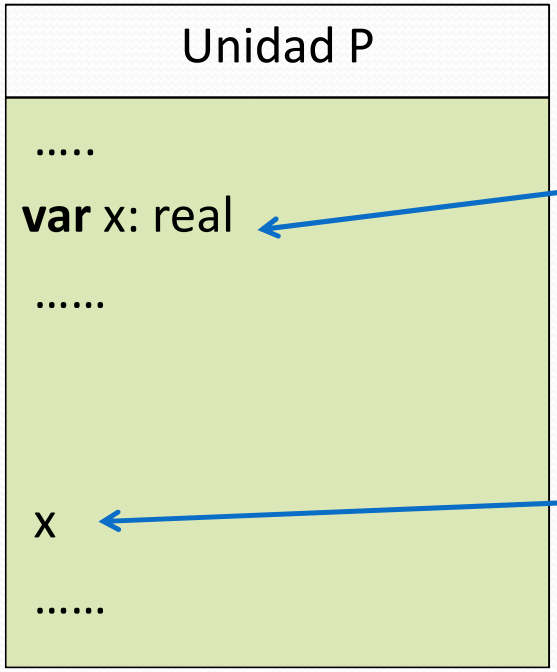


22

Representación en memoria

El acceso a cualquier objeto de dato declarado en P, se calcula como:

Dirección Registro de Activación P + desplazamiento de la variable



En la declaración se determina el desplazamiento de la variable x

En la uso se accede a la misma mediante la fórmula:

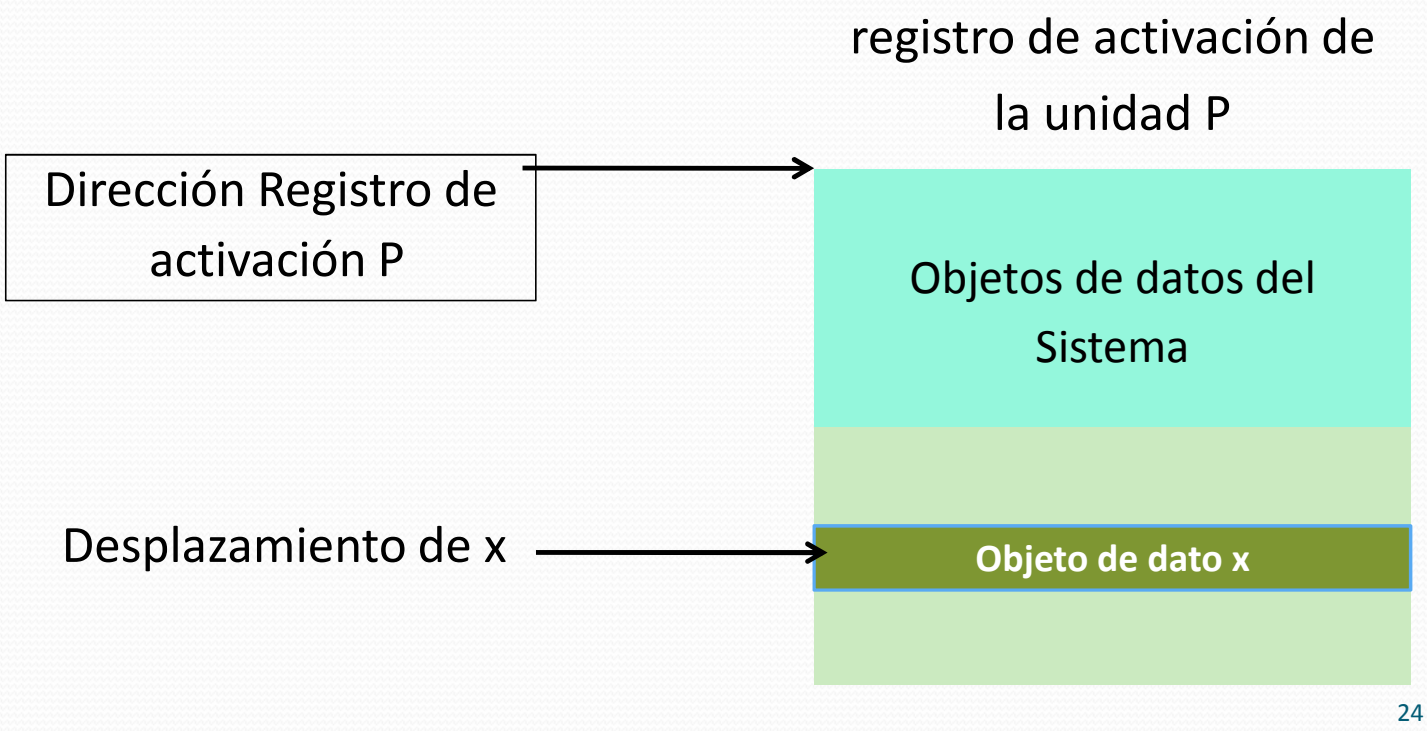
Dir. RA P + desplazamiento de x

23

Representación en memoria

El acceso a cualquier objeto de dato declarado en P, se calcula como:

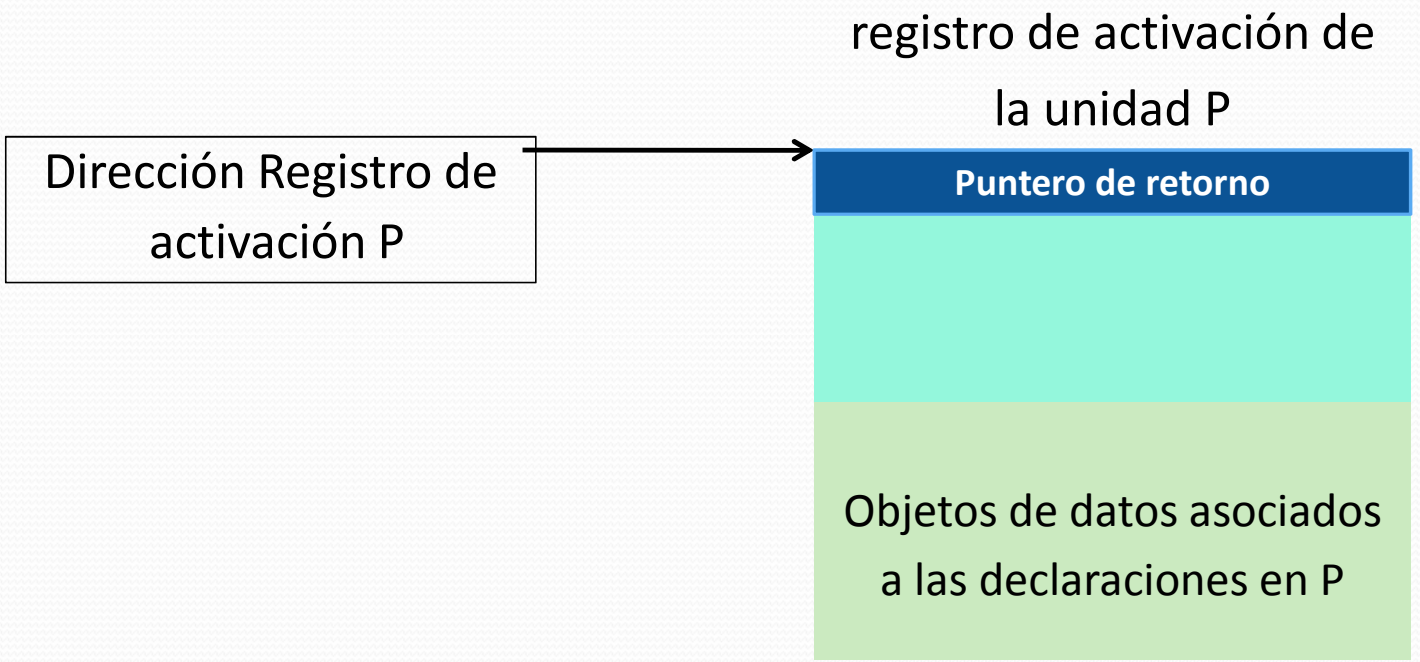
Dirección Registro de Activación P + desplazamiento de la variable



24

Representación en memoria

Si tratamos con unidades simples, sin parámetros, el único objeto del sistema que resulta imprescindible guardar es el puntero de retorno.



25

Representación en memoria

Los lenguajes de programación puede ser clasificados de acuerdo las decisiones que toman en diseño. Se puede clasificar de acuerdo al manejo de memoria y a las regla de alcance.

La clasificación de acuerdo al *manejo de memoria* que utilizan no conduce a lenguajes:

- Estático
- Basado en pila
- Dinámico

26

Representación en memoria

Por otro lado la clasificación de acuerdo a las reglas de alcance y al esquema de manejo de memoria:

- Alcance estático
  - Alocación estática
  - Alocación semiestática (basada en pila)
  - Alocación semidinámica (basada en pila)
  - Alocación dinámica (basada en pila)
- Alcance dinámico

Finalmente si tomamos en cuenta la resolución de invocaciones a objetos de datos

- Acceso local
- Acceso global
- Esquema de alocación

27

LP con Alcance Estático–Alocación Estática

En **compilación** se determina:

- El alcance de cada variable y el ambiente de referenciamiento de cada unidad.
- El espacio requerido para mantener cada variable local a la unidad y su desplazamiento dentro del registro de activación.
- El espacio requerido para ejecutar la unidad.
- La asociación entre cada variable local del código y cada objeto de dato del registro de activación a través del par: *dirección base del registro de activación + desplazamiento*.
- La asociación entre cada variable global del código y cada objeto de dato en el área compartida: *dirección base del área compartida + desplazamiento*.

28

LP con Alcance Estático–Alocación Estática

En **vinculación** se conoce el desplazamiento de:

- La primera instrucción de cada unidad en el área de código C, respecto a una dirección base dC.
- Cada registro de activación en el área de datos, respecto a una dirección base dD.

En la **carga** del programa:

- Se reserva espacio de almacenamiento para todo el código y todos los registros de activación.
- Cada segmento de código queda asociado a un único registro de activación.

29

LP con Alcance Estático–Alocación Estática

Acceso al entrono local:

Programa
program main integer i,j i:= 1 k:= 1 j:= i + k

Desplazamiento	Identificador	Tipo
0	i	Integer
1	j	Integer
2	k	Integer

Observaciones:

- k está declarada en forma implícita.
- El desplazamiento 0 corresponde al puntero de retorno.

30

LP con Alcance Estático–Alocación Estática

Acceso al entrono local:

Programa
program main integer i,j i:= 1 k:= 1 j:= i + k

Segmento de código
SET Main+,1
SET Main+2,1
SET Main+1, D[Main+0]+D[Main+2]

31

LP con Alcance Estático–Alocación Estática

Acceso al entrono local:

Unidad
subroutine P integer i,j i:= 1 k:= 1 j:= i + k

Segmento de código
SET dP+1,1
SET dP+3,1
SET dP+2, D[dP+1]+D[dP+3]

	Registro de activación
0	Puntero de retorno
1	i
2	j
3	k

32

LP con Alcance Estático–Alocación Estática

Acceso al entrono global:

Programa
subroutine unit integer i,j **common k** i:= 1 k:= 1 j:= i + k

Desplazamiento	Identificador	Tipo
1	i	Integer
2	j	Integer
0	**k**	Integer

33

LP con Alcance Estático–Alocación Estática

Acceso al entrono global:

Programa
subroutine unit integer i,j **common k** i:= 1 k:= 1 j:= i + k

Segmento de código
SET unit+1,1
SET **Common**+0,1
SET unit+2, D[unit+1]+D[**Common**+0]

34

LP con Alcance Estático–Alocación Estática

Acceso al entrono global:

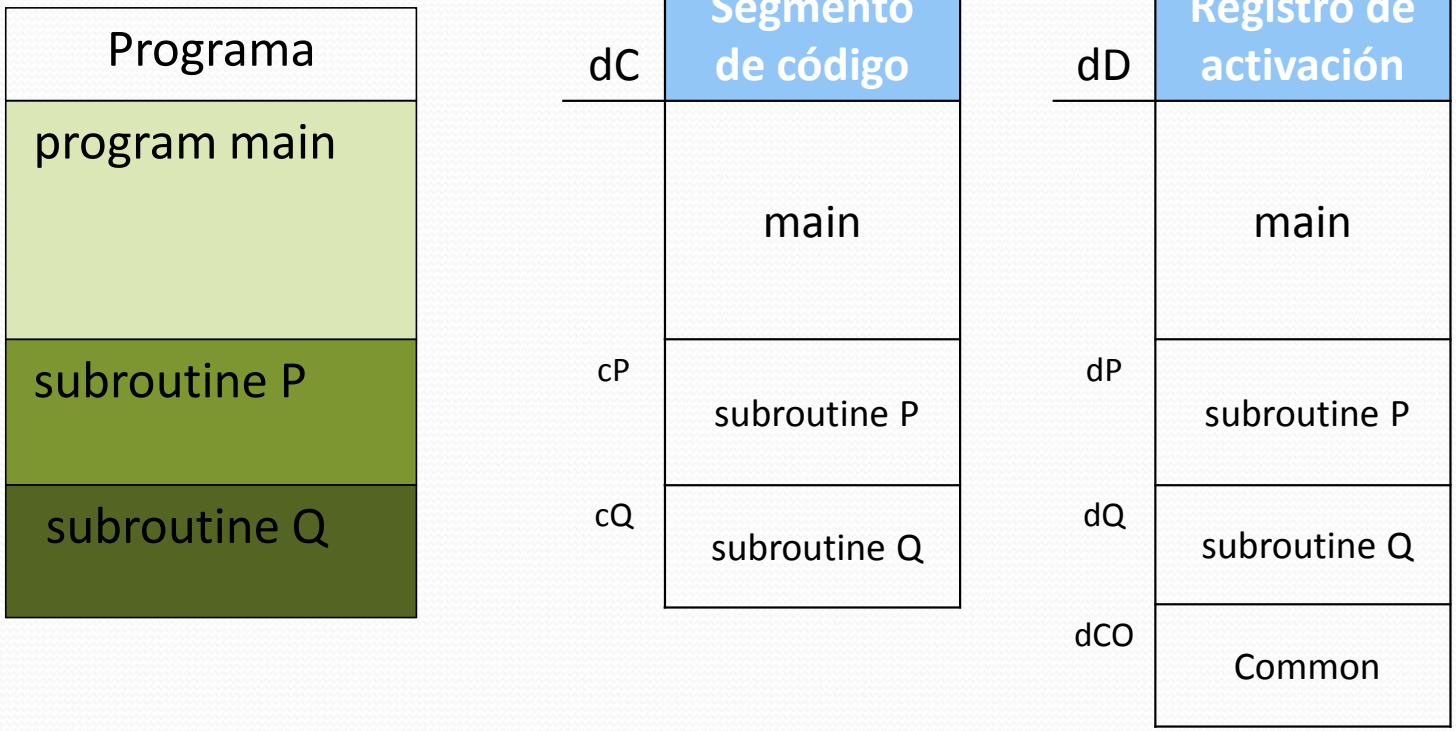
Programa
subroutine unit integer i,j **common k** i:= 1 k:= 1 j:= i + k

Segmento de código
SET unit+1,1
SET **Common+0** ,1
SET unit+2, D[unit+1]+D[**Common**+0]

	Registro de activación
0	k
1	puntero de retorno
2	i
3	j

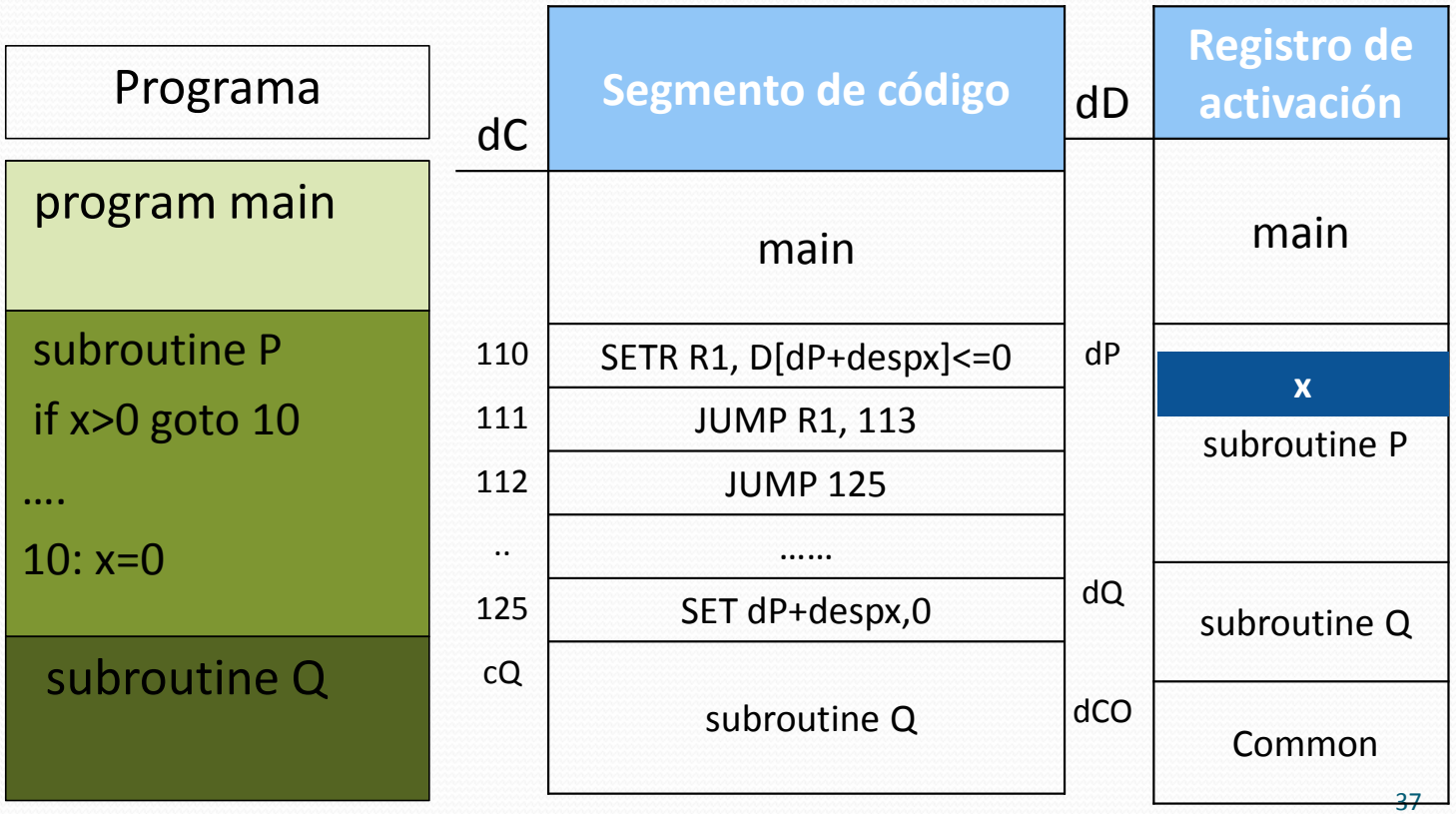
35

LP con Alcance Estático–Alocación Estática

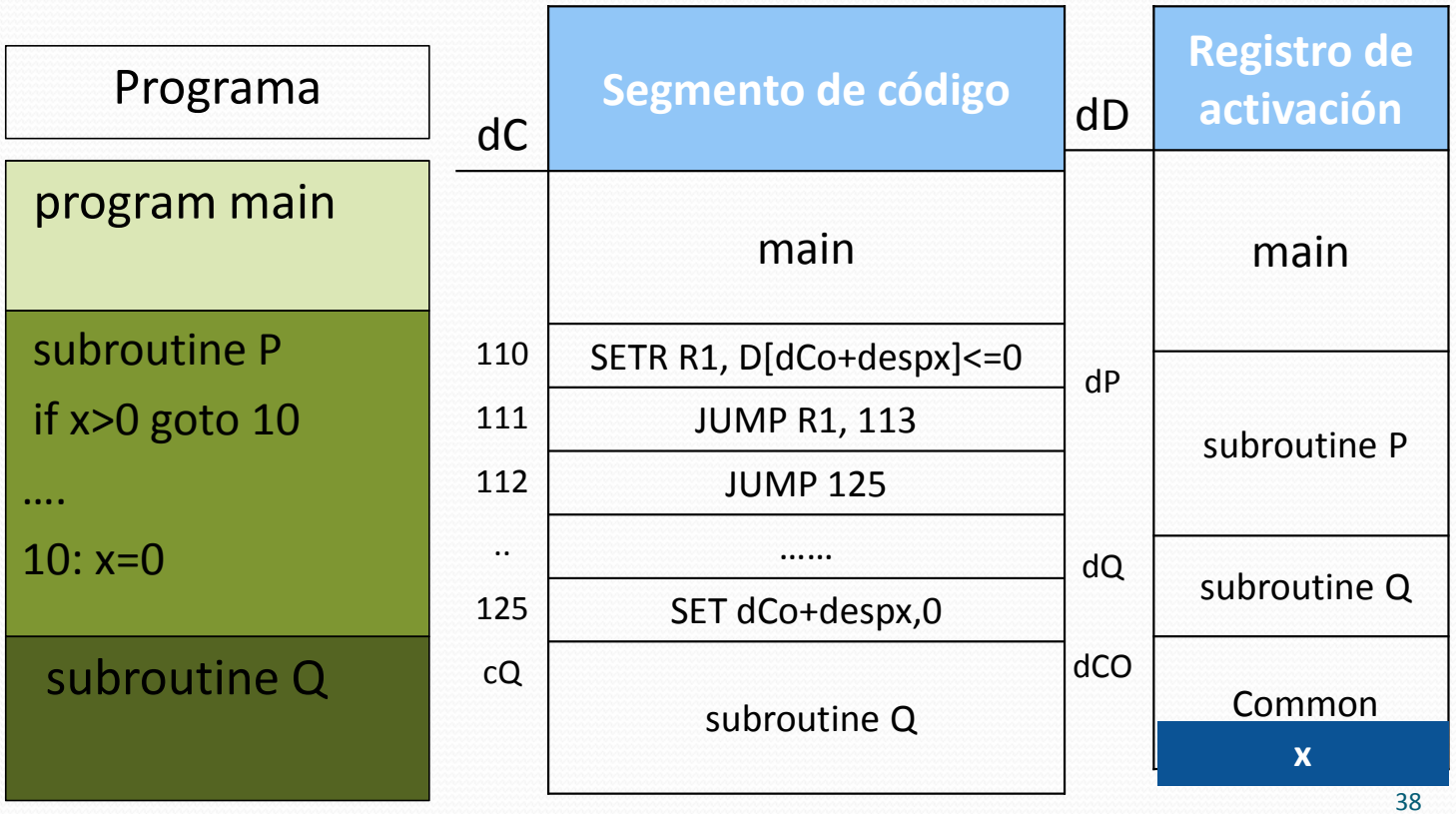


36

LP con Alcance Estático–Alocación Estática



LP con Alcance Estático–Alocación Estática



LP con Alcance Estático–Alocación Estática

Activación de una unidad, pasos a realizar:

- Guardar el puntero de retorno en el primer objeto de dato del registro de activación de la unidad llamada.
- Actualizar el IP con la dirección de la primera instrucción de la unidad llamada.

La traducción involucra estas dos instrucciones del procesador abstracto:

- SET dU,IP+2 (se guarda en área de datos una dirección del área de código)
- JUMP cU.

39

LP con Alcance Estático–Alocación Estática

Retorno de una unidad, paso a realizar:

- Transferir el control recuperando el puntero de retorno almacenado en el primer objeto de dato del registro de activación de la unidad llamada.

La traducción involucra esta instrucción del procesador abstracto:

- JUMP D[dU].

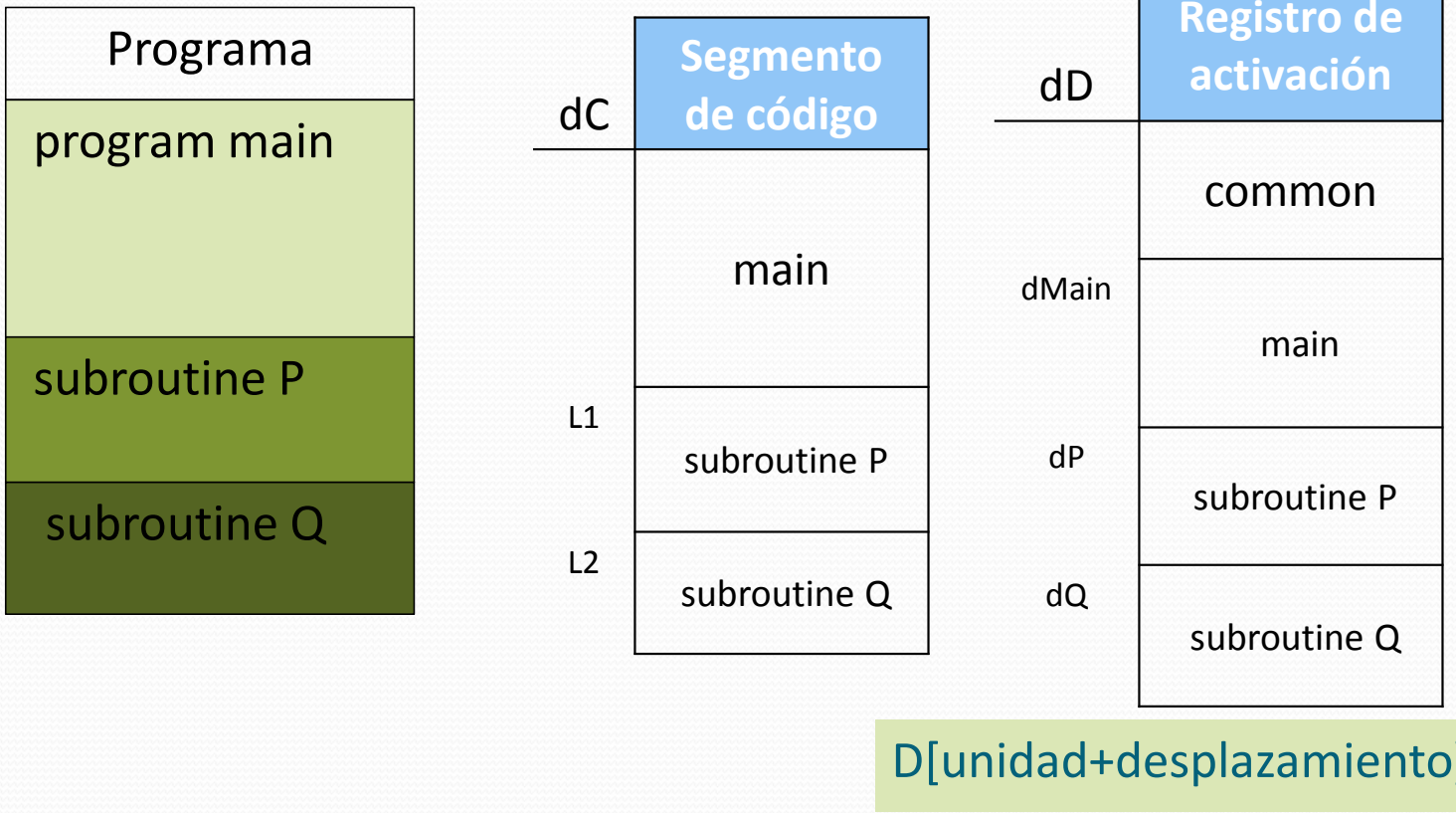
LP con Alcance Estático–Alocación Estática

Ejemplo, sea el siguiente programa:

Programa	Unidad P	Unidad Q
Program main	Subprogram P	Subprogram Q
integer i,j	integer i,k	integer i,m,n
common x,y	common x,y	common x,y
call P	call Q	x = m*n
j = i*x	…..	….
…..	End	End
End		

41

LP con Alcance Estático–Alocación Estática



D[unidad+desplazamiento]

42

LP con Alcance Estático–Alocación Estática

Ejemplo

```
Program main
integer i,j
common x,y

call P
j = i*x
.....
End
```

Identificador	Desplaz.	Unidad
i	0	main
j	1	main
x	0	common
y	1	common

43

LP con Alcance Estático–Alocación Estática

Ejemplo

Programa
Program main
integer i,j
common x,y
call P
j = i*x
…..
End

Segmento de código
SET dP, ip+2
JUMP L1
SET Main+2, D[Main+1]+D[Common+0]

	Registros de activación
0	x
1	y
0	Ptr. Ret
1	i
2	j

44

LP con Alcance Estático–Alocación Estática

Ejemplo

Unidad P
Subprogram P
integer i,k
common x,y
call Q
…..
End

Identificador	Desplaz.	Unidad
x	0	common
y	1	common
i	1	P
k	2	P

LP con Alcance Estático–Alocación Estática

Ejemplo

Unidad P
Subprogram P
integer i,k
common x,y
call Q
…..
End

Segmento de código
SET dQ, ip+2
JUMP L2
……
JUMP D[dP+0]

	Registro de activación de P
0	Ptr. Ret
1	i
2	k

46

LP con Alcance Estático–Alocación Estática

Ejemplo

```
Unidad Q
Subprogram Q
 integer i,m,n
 common x,y

 x = m*n
 ....
End
```

Identificador	Desplaz.	Unidad
x	0	common
y	1	common
i	1	Q
m	2	Q
N	3	Q

LP con Alcance Estático–Alocación Estática

Ejemplo

Unidad Q
Subprogram Q
integer i,m,n
common x,y
x = m*n
….
End

Segmento de código
SET common+0, D[dQ+2]*d[dQ+3]
….
……
JUMP D[dQ+0]

	Registro de activación de Q
0	Ptr. Ret
1	i
2	m
3	n

48

Alcance Estático–Alocación basada en pila

En **compilación**:

- Se determina la distancia entre la unidad que rereferencia una variable y la unidad que contiene la declaración.
- No es posible anticipar el espacio de memoria requerido para ejecutar el programa.

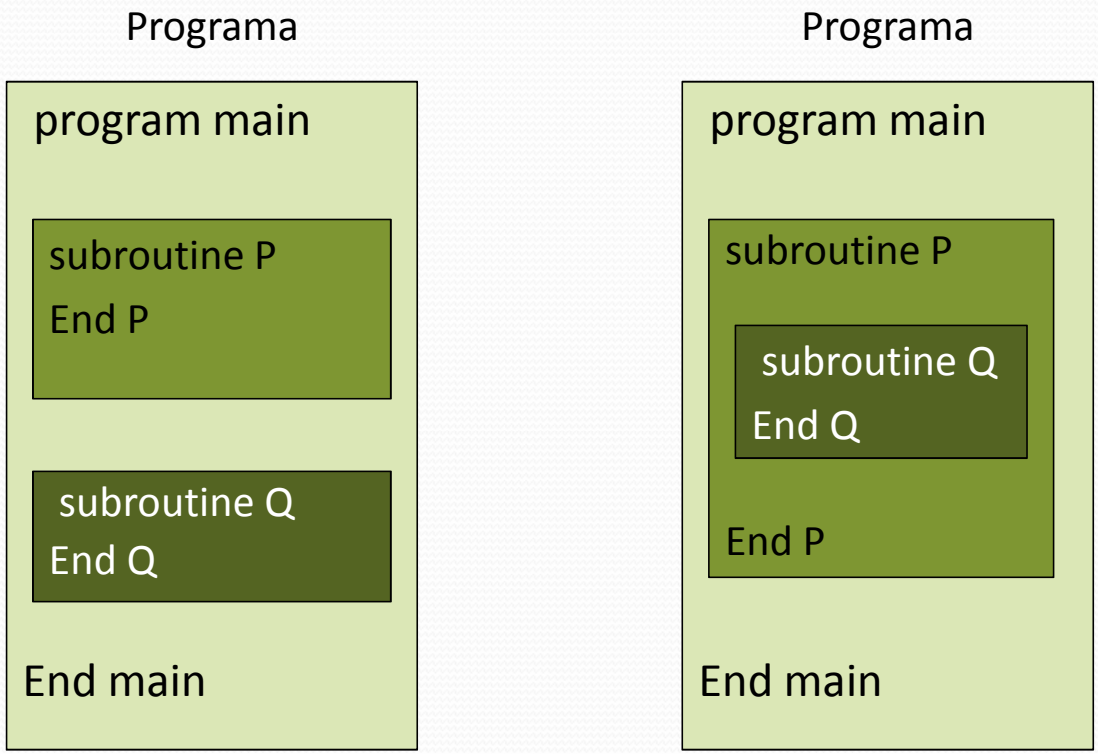
Alcance Estático–Alocación basada en pila

En **ejecución**:

- Se reserva espacio de almacenamiento para el registro de activación en el momento en que la unidad se activa.
- Se conoce la dirección base de cada registro de activación.
- Los registros de activación ocupan y liberan memoria siguiendo un comportamiento de pila.
- Cada segmento de código puede quedar asociado a varios registros de activación

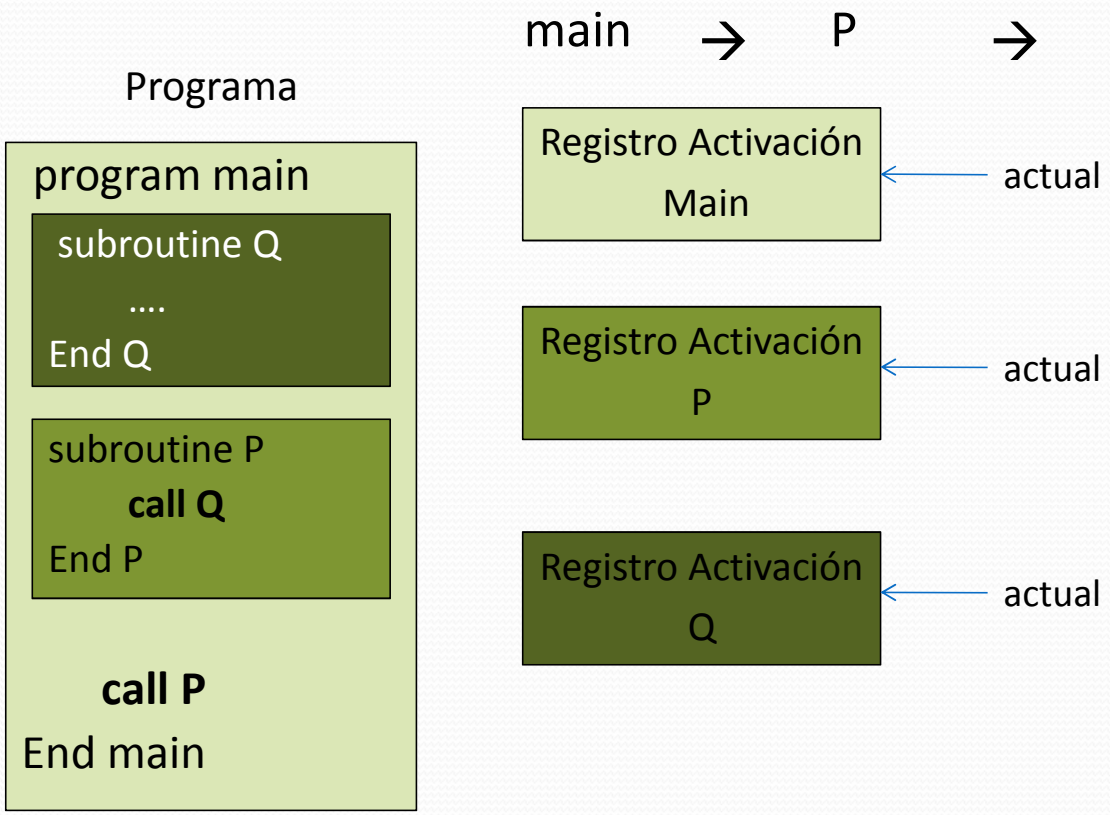
50

Alcance Estático–Alocación basada en pila



51

Alcance Estático–Alocación basada en pila



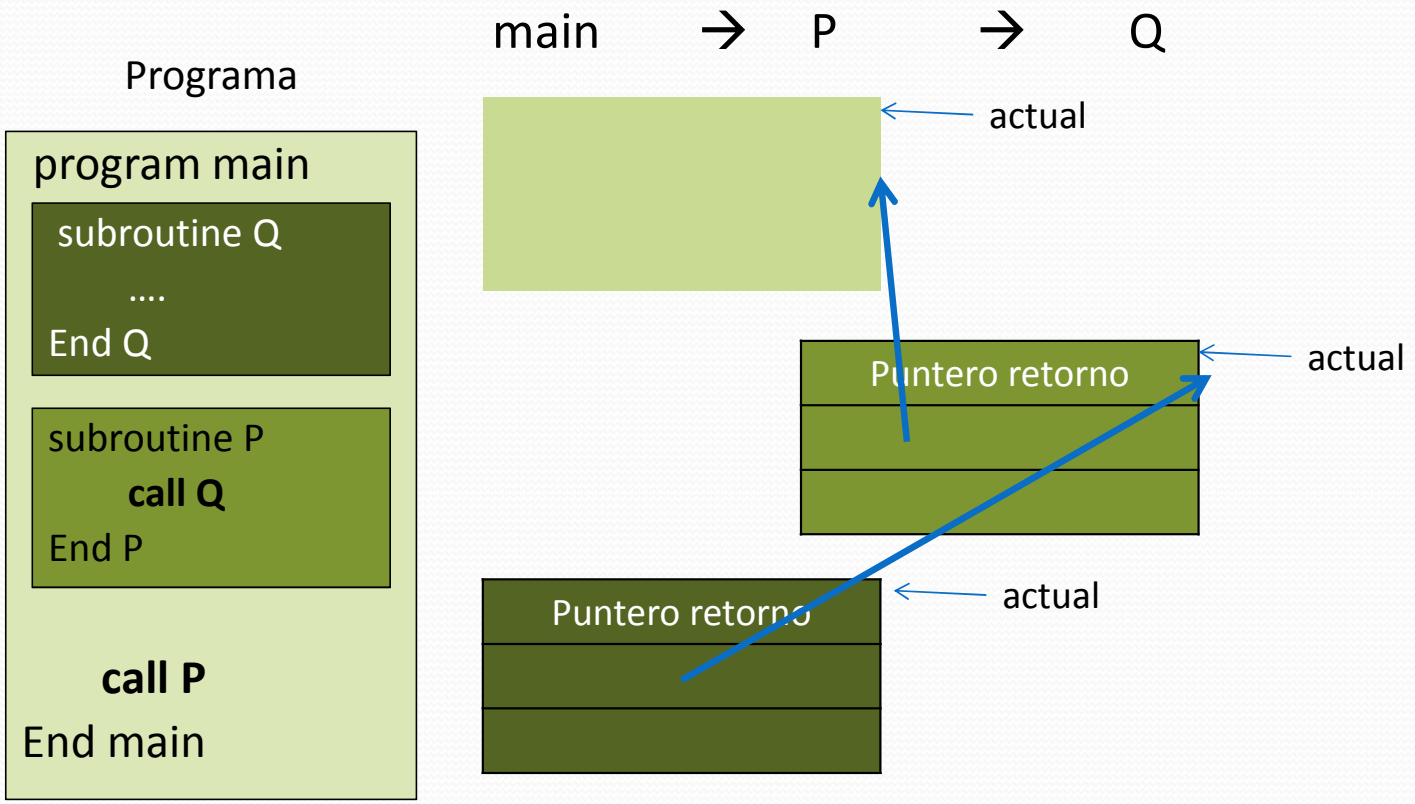
52

Alcance Estático–Alocación basada en pila

Resulta necesario mantener la cadena dinámica de llamadas, en orden de poder resolver las referencias no locales. Es por ello que el registro de activación necesita mantener más información que en el modelo anterior.

Registro de activación de la unidad
puntero de retorno
enlace dinámico

Alcance Estático–Alocación basada en pila



54

Alcance Estático–Alocación basada en pila

Activación de una unidad, pasos a realizar:

- Crear el registro de activación
- Guardar puntero de retorno
- Guardar enlace dinámico
- Actualizar actual
- Actualizar IP

55

Alcance Estático–Alocación basada en pila

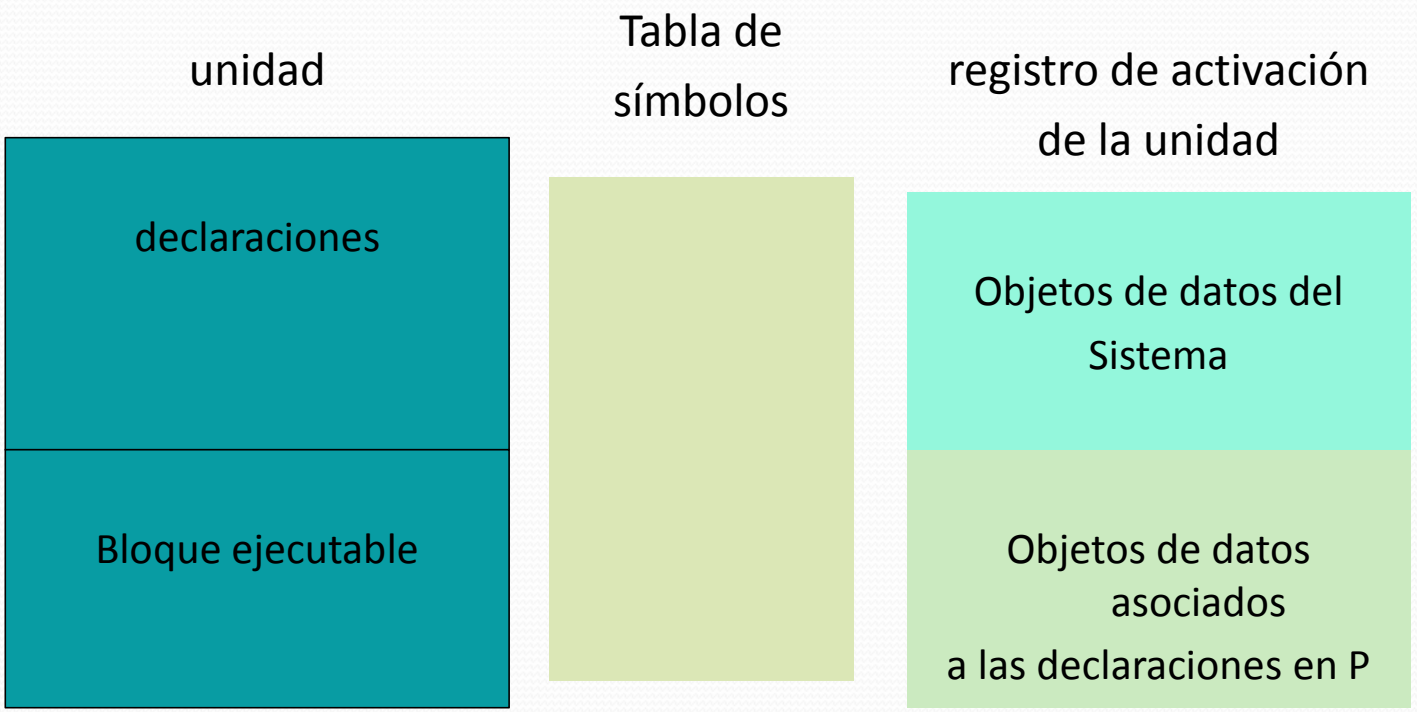
Retorno de una unidad, paso a realizar:

- Actualizar actual utilizando el enlace dinámico.
- Actualizar IP utilizando el puntero de retorno.
- Destruir el registro de activación

Este modelo admite **Recursividad**. En caso de utilizarlo, en la cadena dinámica aparecen más de un registro de activación asociados a la misma unidad.

56

Alcance Estático–Alocación basada en pila



57

Alcance Estático–Alocación basada en pila

Acceso al entorno local – estructura de bloques anidados

```
Program main
Var i,j: real;
 Procedure q
 var j,k,l: real;
 ....
 end q
....
End main
```

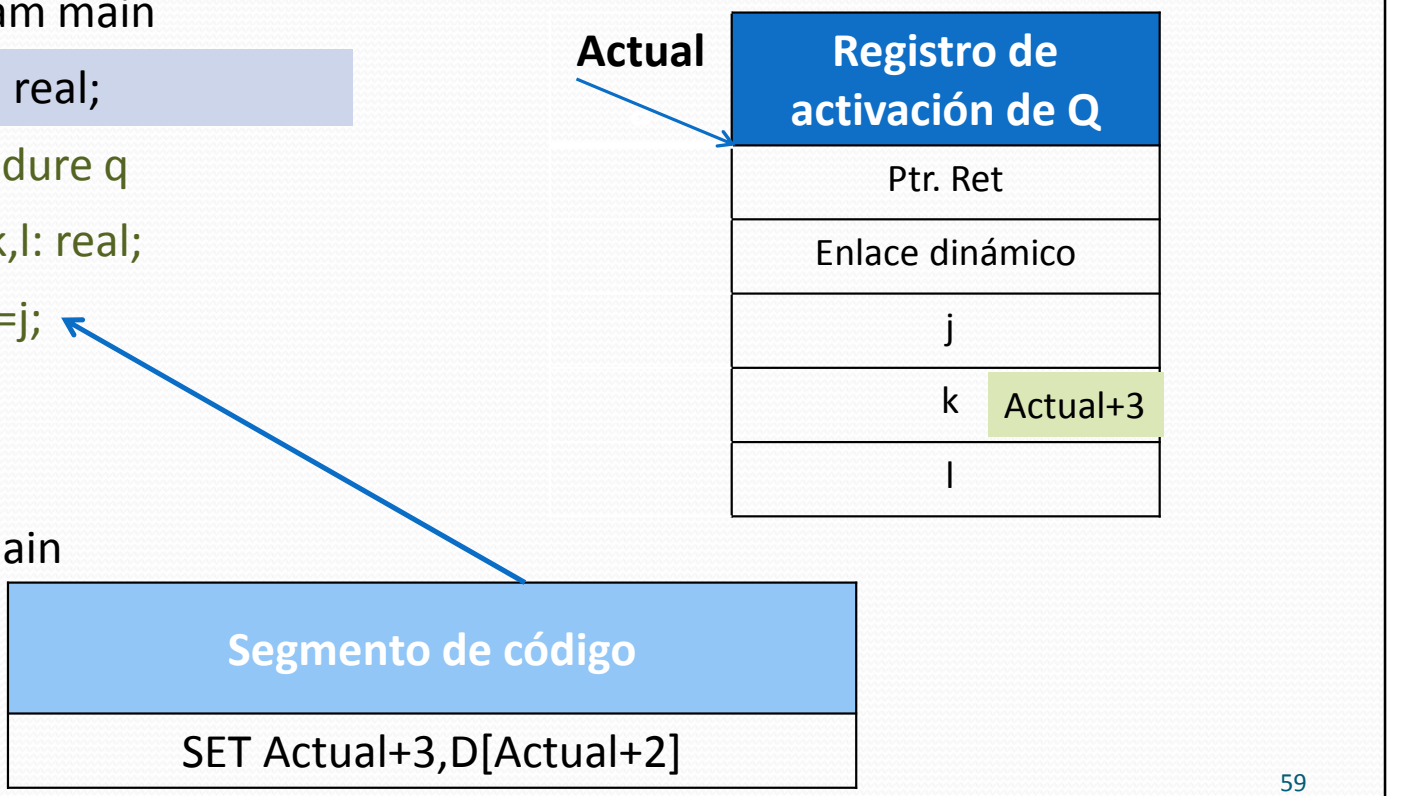
Identificador	Tipo	Anid.	Desplaz.
i	real	0	0
j	real	0	1
j	real	1	2
k	real	1	3
l	real	1	4

58

Alcance Estático–Alocación basada en pila

Acceso al entorno local – estructura de bloques anidados

```
Program main
Var i,j: real;
 Procedure q
 var j,k,l: real;
 …., k:=j;
 end q
….
End main
```



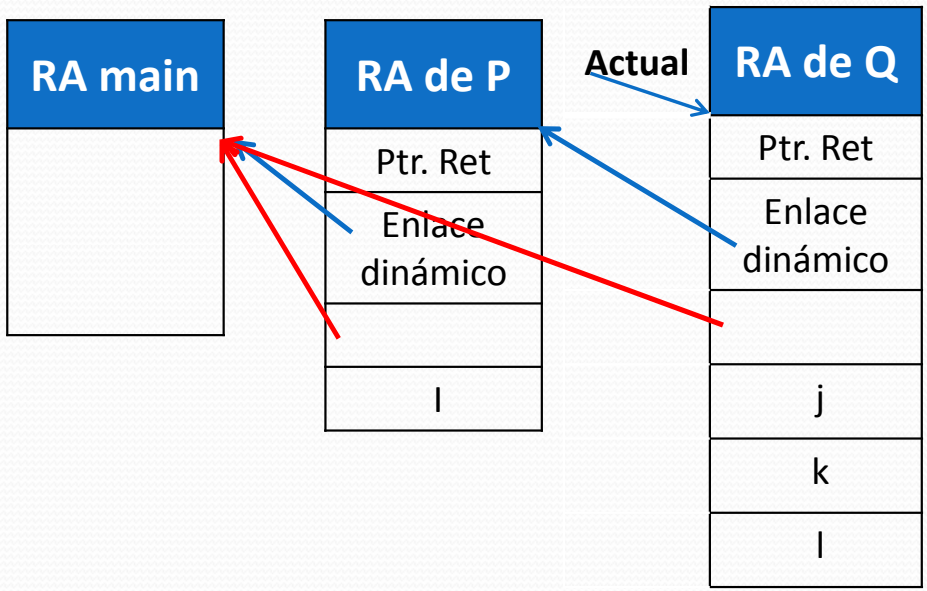
59

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados

```
Program main
.....
 Procedure q
 var j,k,l: real;
 .... i
 end q
Procedure p
Var i: real;
     .... i
End P
End main
```

main → P → Q



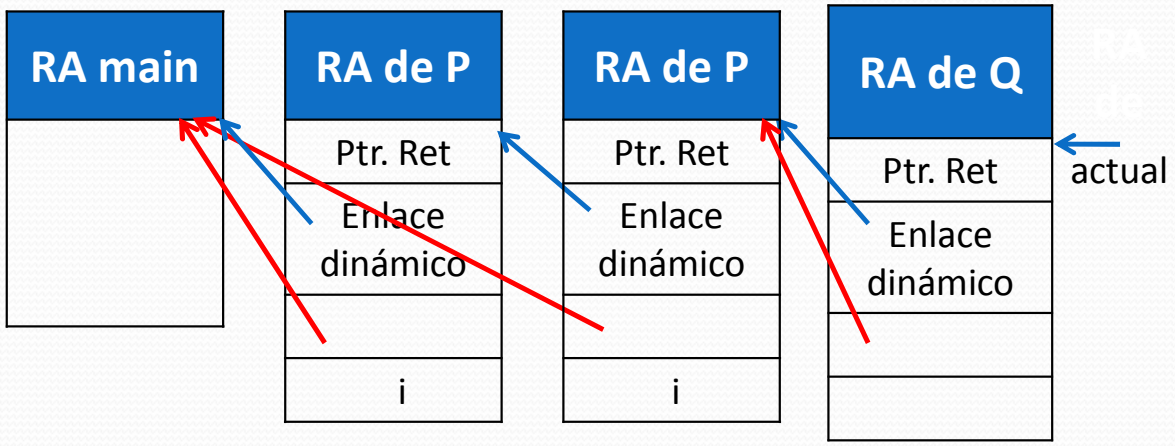
60

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados - recursividad

```
Program main
 Procedure p
 var i: real;
     Procedure q
          i
     End q
End p
End main
```

main → P → P → Q



61

Alcance Estático–Alocación basada en pila

Si se resuelven los accesos no locales, en lenguajes con estructuras de bloques anidados, se debe mantener información sobre el enlace estático de la unidad. Una forma de hacerlo, es manteniéndolo en el registro de activación de la unidad

Registro de activación de la unidad
puntero de retorno
enlace dinámico
enlace estático

62

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados

```
Program main
Var i,j: real;
 Procedure q
 var j,k,l: real;
      i
 end q
....
End main
```

Identificador	Tipo	Anid.	Desplaz.
i	real	0	0
j	real	0	1
j	real	1	3
k	real	1	4
l	real	1	5

La distancia entre el uso y el nivel de anidamiento de la declaración es 1

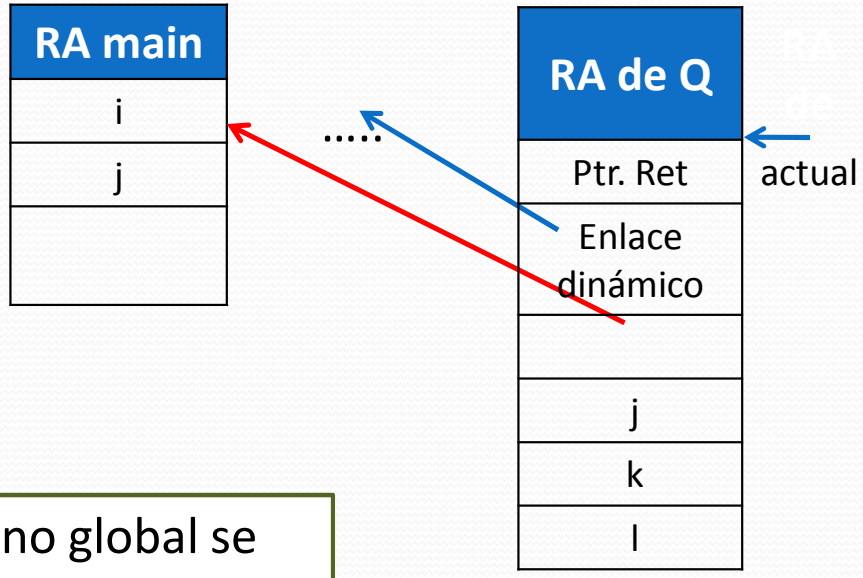
63

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados

```
Program main
Var i,j: real;
 Procedure q
 var j,k,l: real;
      i
 end q
....
End main
```

main → P → Q



El acceso al entorno global se logra con **D[Actual+2]**

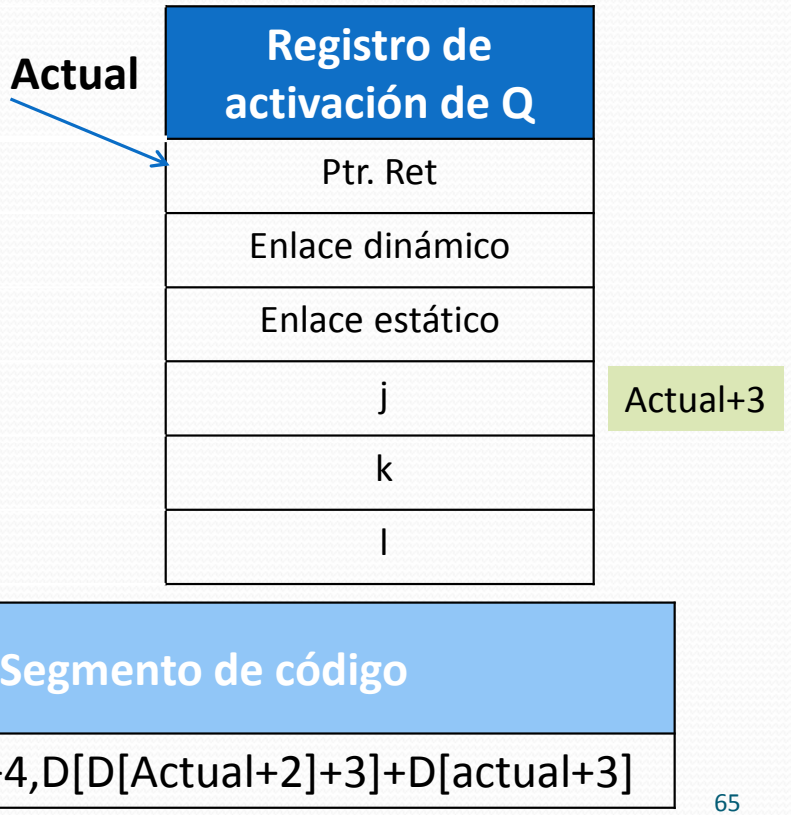
El acceso a la variable i con **D[D[Actual+2]+0]**

64

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados

```
Program main
Var i,j: real;
 Procedure q
 var j,k,l: real;
 …., k:=i+j;
 end q
….
End main
```



65

Alcance Estático–Alocación basada en pila

Acceso al entorno global – estructura de bloques anidados

Si la distancia es uno, como en el ejemplo, la situación se resuelve con una indirección; sin embargo claramente la distancia puede ser mayor.

La cantidad de indirecciones involucradas es igual a la distancia entre el uso y la declaración. A mayor cantidad de indirecciones, mayor es el costo

66

Alcance Estático–Alocación basada en pila

Alocación de memoria

Para trabajar con la pila de datos y tener el control de los registros de activación vamos a contar con dos registros rápidos:

- actual: mantiene una referencia al comienzo del registro de activación correspondiente a la unidad que se encuentra en ejecución.
- libre: mantiene una referencia al área de datos a partir de donde la misma está disponible para su uso.

La alocación de memoria sigue un comportamiento de pila

Alcance Estático–Alocación basada en pila

Alocación semi-estática de memoria

En **compilación** se determina:

- El espacio requerido para mantener cada variable local a una unidad.
- El tamaño del registro de activación
- El desplazamiento de cada variable local dentro del registro de activación.
- La distancia entre la unidad que referencia a una variable y la unidad que contiene la declaración.

68

Alcance Estático–Alocación basada en pila

Alocación semi-estática de memoria

En **compilación** se determina:

- La asociación entre cada variable local del código y cada objeto de dato del registro de activación a través del par: [dirección base RA + desplazamiento].
- La asociación entre cada variable global del código y cada objeto de dato en la Pila. La dirección base considera la distancia entre la declaración y la referencia, utilizando la cadena estática.

69

Alcance Estático–Alocación basada en pila

Activación de una unidad

a. **Guardar el puntero de retorno**:

se realiza a través de la siguiente instrucción

Pila[libre,0] ← IP + T

Siendo T la cantidad de instrucciones que se van a saltear dentro de la unidad llamadora (como mínimo hay que saltear la instrucción JUMP que realiza la transferencia de control, si hay pasaje de parámetros hay que saltear las instrucciones vinculadas y otras)

b. **Guardar enlace dinámico**:

Pila[libre,1] ← actual

70

Alcance Estático–Alocación basada en pila

Activación de una unidad

c. **Guardar el enlace estático**:

se realiza a través de la siguiente instrucción

Pila[libre,2] ← expresión de cálculo de EE

Analizaremos la expresión de cálculo de EE a contiuación.

d. **Actualizar actual**:

actual ← libre

e. **Actualizar libre**:

libre ← libre + S

Siendo S el tamaño del registro de activación (objetos de datos del sistema, parámetros y variables locales). Este tamaño es calculado por el compilador

71

Alcance Estático–Alocación basada en pila

f. **Actualizar IP**:

se realiza a través de la siguiente instrucción

IP ← dirección del segmento de código de la unidad invocada (Cc)

La traducción requiere:

a. **Guardar el puntero de retorno**
b. **Guardar enlace dinámico**
c. **Guardar enlace estático**
d. **Actualizar actual**
e. **Actualizar libre**
f. **Actualizar IP**

72

Cálculo del enlace estático

```
Program Principal
   Procedure Q
      Procedure R
      end R
   end Q
   Procedure P
   End P
End Principal
```

Hay tres situaciones posibles:

1. Dos unidades (diferentes entre si) P y Q, ambas del mismo nivel léxico donde P → Q.
2. Una unidad, Q, invoca a otra que está declarada dentro de ella, R.

   Q → R
3. Una unidad, R, invoca a una unidad que la contiene.

   R → Q

Cálculo del enlace estático

```
Program Principal
   Procedure Q
      Procedure R
      end R
   end Q
   Procedure P
   End P
End Principal
```

Situación 1: P → Q.

En este caso, la distancia entre la unidad llamadora y la llamada es igual a cero(ambas tienen el mismo nivel léxico) por lo tanto en enlace estático de P es el enlace estático de Q también.

Enlace estático de Q ← PILA[ACTUAL,2]

Cálculo del enlace estático

```
Program Principal
   Procedure Q
      Procedure R
      end R
   end Q
   Procedure P
   End P
End Principal
```

Situación 2: Q → R.

En este caso, la distancia entre la unidad llamadora y la llamada es igual a uno por lo tanto en enlace estático de R es la dirección del registro de activación de la unidad llamadora , es decir Q.

Enlace estático de R ← ACTUAL

Cálculo del enlace estático

```
Program Principal
    Procedure Q
        Procedure R
        end R
    end Q
    Procedure P
    End P
End Principal
```

Situación 3: R → Q.

En este caso, la distancia entre la unidad llamadora y la llamada es igual a negativa, -1. en este caso debe existir una instancia de Q suspendida anteriormente, ya que no habría instancia de R sin alguna instancia de Q. existe por lo tanto una forma de recursión cruzada involucrada.

Se debe recorrer la cadena estática tantos niveles como intervienen en la distancia

Enlace estático de Q ← PILA[PILA[ACTUAL,2],2]