Lenguajes de Programación

Dr. Carlos A. Coello Coello
Departamento de Computación
CINVESTAV-IPN
ccoello@cs.cinvestav.mx

Implementación de los Lenguajes Estructurados

- En la práctica, los programas suelen acceder más frecuentemente a las variables locales y a las globales (o sea, a las variables declaradas en los procedimientos más internos y más externos).

- Por tanto, en programas que tienen muchos niveles de anidamiento, el tiempo promedio para acceder a una variable puede ser muy alto. Esto es importante, porque el tiempo requerido para acceder las variables suele dominar el tiempo de ejecución de un programa.

Implementación de los Lenguajes Estructurados

- En FORTRAN, el estado de cada subprograma se representa en un registro de activación, el cual mantiene toda la información necesaria para caracterizar el estado de la operación en progreso.

- Esto incluye el almacenamiento para los parámetros del procedimiento, sus variables locales y temporales, la dirección de regreso del subprograma y una *liga dinámica* (esto es un apuntador al registro de activación del invocador).

Implementación de los Lenguajes Estructurados

- Estos registros de activación pueden adaptarse fácilmente a los lenguajes estructurados, que permiten recursividad.

- Combinar las necesidades de los accesos ambientales bajo la invocación y el retorno de una estructura de bloques y de un procedimiento recursivo, implica que el registro de activación de un procedimiento tiene los siguientes componentes:

Implementación de los Lenguajes Estructurados

1. La **EP** (parte del ambiente), que define el contexto a ser utilizado por esta activación del procedimiento y que contiene lo siguiente:

A) Los *parámetros* y las variables más internas (*locales*).

B) La *liga estática*, que proporciona acceso al ambiente circundante (no local).

Implementación de los Lenguajes Estructurados

2. La **IP** (parte de la instrucción), que designa la instrucción actual que está siendo (o está por ser) ejecutada en esta activación del procedimiento.

3. La *liga dinámica*, que apunta al registro de activación del invocador de esta activación del procedimiento y que nos permite restaurar el estado del invocador tras la salida del procedimiento.

Implementación de los Lenguajes Estructurados

- Una pregunta que puede surgir a este punto es: ¿podemos combinar las dos ligas antes mencionadas (la estática y la dinámica) en una sola? Después de todo, ambas apuntan a registros de activación.

- Resulta que sí se necesitan ligas separadas.

- Recordemos que Pascal y Algol usan reglas de entorno estáticas. Esto es, un procedimiento se ejecuta en el ambiente en que se define y no en el ambiente de su invocador.

Implementación de los Lenguajes Estructurados

- En estos lenguajes con reglas de entorno estático, el campo de la *liga dinámica* en el registro de activación de un procedimiento, apunta al registro de activación anterior en la pila, el cual es el registro de activación del invocador.

- Por lo tanto, si seguimos la cadena dinámica, no obtendremos el ambiente correcto del procedimiento. Necesitamos obtener el ambiente del invocador en vez del ambiente de definición.

Implementación de los Lenguajes Estructurados

- Además, el ambiente de definición no se encuentra a una posición fija en la cadena dinámica.

- Para ilustrar esto, supongamos que tenemos dos procedimientos P y Q, que se definen en el mismo procedimiento B y que P se invoca a sí mismo recursivamente varias veces antes de invocar a Q.

- Cuando Q ha sido invocado, la pila lucirá como en la imagen que se mostrará en clase.

Implementación de los Lenguajes Estructurados

- Cabe destacar que el ambiente de definición de Q es B. No hay manera de que el compilador pueda saber cuántos registros de activación de P se encuentran entre Q y B, puesto que este número depende del comportamiento dinámico de P.

- En otras palabras, no hay manera simple de que Q pueda llegar a su contexto B, vía la cadena dinámica.

- Una solución fácil a este problema es proporcionar un apuntador explícito de Q a su ambiente de definición (ver imagen en clase). Este apuntador es la **liga estática**.

Implementación de los Lenguajes Estructurados

- En FORTRAN, para invocar un subprograma, se requieren cuatro pasos:

1. Transmitir los parámetros al (subprograma) invocado.
2. Almacenar el estado del invocador en el registro de activación del invocador.
3. Establecer la liga dinámica del invocado al invocador.
4. Entrar al invocado en su primera instrucción.

Implementación de los Lenguajes Estructurados

- Estos pasos se pueden reorganizar en las siguientes funciones básicas que deben realizarse para activar un procedimiento:

1. El estado del invocador debe almacenarse (paso 2 de los antes indicados).
2. Debe crearse un registro de activación para el invocado (pasos 1 y 3 de los antes indicados).
3. Debe entrarse al invocado en el contexto del nuevo registro de activación (paso 4 de los antes indicados).

Implementación de los Lenguajes Estructurados

- Esto es, para desactivar al invocador y activar al invocado, es necesario:

(1) suspender al invocador en su registro de activación,

(2) inicializar el registro de activación del invocado y

(3) transferir el lugar de control del invocador al invocado.

Implementación de los Lenguajes Estructurados

- Consideremos ahora al primero de estos pasos, donde se almacena el estado del invocador. El estado del invocador tiene dos componentes principales: la parte de la instrucción (IP) y la parte del ambiente (EP).

- La IP es la dirección en la cual el invocador debe continuar su ejecución después de que el invocado termine. Por tanto, debemos almacenar esta dirección en la IP del registro de activación del invocador. Esto es:

```
M[EP].IP := continuar;   almacenar la ubicación donde
                         continuaremos
```

Implementación de los Lenguajes Estructurados

- El registro de la EP siempre apunta al registro de activación del procedimiento activo actualmente (o sea, el invocador).
- El segundo componente del estado del invocador es la parte del ambiente, la cual, a su vez, está compuesta de las variables locales y las no locales.
- El acceso a las variables no locales ya se encuentra almacenado en la liga estática del registro de activación del invocador, de tal forma que no se requiere más trabajo para manejar dichas variables. Las variables locales están contenidas en el registro de activación del invocador, de manera que también están a salvo.

Implementación de los Lenguajes Estructurados

- El segundo de los pasos en la activación de un procedimiento es crear un registro de activación para el invocado, debidamente inicializado, e instalar este registro de activación como el nuevo contexto activo.

- ¿Qué se requiere para lograr esto? Hemos visto que el registro de activación de un procedimiento tiene las partes siguientes:

- PAR Parámetros
- SL Liga estática
- IP Dirección de reinicio
- DL Liga dinámica

Implementación de los Lenguajes Estructurados

- Por lo tanto, cada una de estas partes debe inicializarse debidamente.

- También, para instalar este registro de activación como el nuevo contexto activo, debe colocarse un apuntador al registro de activación en el registro de la EP (el cual apunta al inicio de la cadena estática).

- A continuación, consideraremos cada uno de estos pasos en el orden listado anteriormente, aunque se hace notar que, no necesariamente tienen que llevarse a cabo en ese orden en particular. De hecho, el mejor orden para las operaciones, usualmente depende de la forma específica en que esté organizado el registro de activación, el cual varía, según la computadora.

Implementación de los Lenguajes Estructurados

- Un registro de activación contiene los parámetros para esta activación en particular (el valor de los parámetros, si se pasan por valor, su dirección si se pasan por referencia, o un apuntador a un *thunk*, si se pasan por nombre). A esto se le denomina PAR (*parameter activation record*).

- Por lo tanto, cada parámetro debe ser evaluado para obtener ya sea un valor, o una dirección, según resulte apropiado, y este resultado debe almacenarse en el lugar adecuado del registro de activación.

Implementación de los Lenguajes Estructurados

- Por lo tanto, si el invocado representa la dirección del nuevo registro de activación para el invocado, entonces el proceso de transmisión de parámetros puede escribirse de la forma siguiente:

M[invocado].PAR[1] := evaluación del parámetro 1;
M[invocado].PAR[2] := evaluación del parámetro 2;
⋮
M[invocado].PAR[*n*] := evaluación del parámetro *n*;

Implementación de los Lenguajes Estructurados

- Cuando vemos que algo será evaluado, como se muestra en la secuencia de código anterior, la pregunta que surge es: ¿en qué ambiente debe evaluarse? Puesto que los parámetros que se pasan a un procedimiento están escritos en el contexto del invocador, deben también ser evaluados en el contexto de éste.

- Puesto que los parámetros deben evaluarse en el contexto del invocador, la inicialización de la parte de los parámetros debe realizarse antes de que se instale el nuevo registro de activación (invocado) y se vuelva el nuevo contexto activo (o sea, antes de que se altere el registro EP). Esta es una de las restricciones del orden antes mencionado.

Implementación de los Lenguajes Estructurados

- La liga estática es la siguiente parte del registro de activación de un procedimiento que debe ser inicializada.

- Por definición, la liga estática apunta al ambiente de definición del procedimiento. La pregunta es: ¿Cómo llegamos a este ambiente? Usaremos el mismo mecanismo que vimos antes para acceder a las variables.

- En tiempo de compilación, debemos almacenar en la tabla de símbolos, el nivel de anidamiento estático de cada procedimiento en la definición del mismo.

Implementación de los Lenguajes Estructurados

- Posteriormente, cuando se compile una invocación a un procedimiento, el compilador restará el nivel de anidamiento estático del procedimiento del nivel de anidamiento estático del ambiente desde el que se le invoca.

- Esto nos dará la distancia estática entre el ambiente de invocación y la definición del procedimiento y, por tanto, nos dará la cantidad de ligas que deben recorrerse en la cadena estática.

Implementación de los Lenguajes Estructurados

- Resumiremos las operaciones para inicializar la liga estática en las siguientes instrucciones. Si el procedimiento se define en el contexto actual (o sea, la distancia estática de la invocación a la definición es cero), entonces basta el código siguiente:

```
M[invocado]. SL := EP;      fijar la liga estática a
                            este contexto
```

En este caso, el ambiente de definición es el ambiente del invocador.

Implementación de los Lenguajes Estructurados

- Si la distancia de la llamada a la definición es *sd*>0, entonces es necesario atravesar primero la cadena estática para llegar al ambiente de definición:

AP := M[EP];	atravesar la primera liga estática
(*sd*-1) × AP := M[AP];	atravesar las ligas estáticas restantes
M[invocado].SL := AP;	fijar la liga estática

Nótese la similitud entre este código y el usado para acceder una variable no local.

Implementación de los Lenguajes Estructurados

- La inicialización del resto del registro de activación del invocado es muy simple. No hay razón para tener que inicializar este campo en este momento, puesto que sólo será utilizado cuando (y si) el invocado se vuelva invocador, al llamar a un procedimiento.

- El campo de la liga dinámica (DL) es también simple de inicializar. Puesto que apunta al registro de activación del invocador, que está contenido en el registro del EP, es suficiente el código suficiente:

```
M[invocado].DL := EP;        establecer la liga dinámica
```

Implementación de los Lenguajes Estructurados

- El siguiente paso es hacer que el registro de activación del invocado sea el nuevo contexto activo. Puesto que el registro del EP siempre apunta al registro de activación del contexto activo (el procedimiento activo en este momento), esto se logra usando:

```
EP := invocado;      instalar nuevo registro
                     de activación
```

Implementación de los Lenguajes Estructurados

- Los pasos finales son asignar espacio para el registro de activación en la pila y entrar al invocado (en su primera instrucción):

SP := SP + tamaño(RA del invocado)

goto entrada(invocado); entrar al invocado

RA = Registro de Activación

Tanto el tamaño del registro de activación del invocado como la dirección del punto de entrada del invocado, son constantes que el compilador conoce.

Implementación de los Lenguajes Estructurados

- La última instrucción en el código antes mostrado, se usa para almacenar información en el registro del IP (*instruction pointer*).
- En otras palabras, '**goto** entrada(invocado)' es equivalente a: 'IP := entrada(invocado)'.
- La transferencia al lugar de control se efectúa mediante las asignaciones a IP y EP (es decir, 'EP := invocado' e 'IP := entrada(invocado)'.
- Los pasos para invocar un procedimiento en Pascal, se resumen en la figura que se mostrará en clase.

Implementación de los Lenguajes Estructurados

- El código mostrado en clase ha sido mejorado ligeramente. Puesto que el registro de EP ya ha sido almacenado, podemos recorrer la cadena estática con *sd* repeticiones de 'EP := M[EP]'.

- Adicionalmente, como SP apunta a la siguiente posición disponible en la pila, puede usarse como la base del registro de activación del invocado, de tal manera que: invocado = SP.

- Por lo tanto, una invocación a procedimiento requiere *sd*+3 referencias de memoria. Nuevamente puede verse que es relativamente caro invocar procedimientos globales desde procedimientos muy anidados (algo bastante común).

Implementación de los Lenguajes Estructurados

- El código para regresar de un procedimiento debe revertir los efectos de su invocación. Esto es, el lugar de control debe ser transferido de vuelta del invocado al invocador. En otras palabras, el invocado debe ser *desactivado* y el invocador debe ser *reactivado*.

- Un retorno de procedimiento es generalmente más simple que una invocación puesto que nos deshacemos de cosas en vez de crearlas. Las dos tareas a realizarse son:

    1. Borrar el registro de activación del invocado.
    2. Restaurar el estado del invocador.

Implementación de los Lenguajes Estructurados

- En la práctica, estos dos pasos deben intercalarse, puesto que la información requerida para restaurar el estado del invocador (o sea, la *liga dinámica*) se encuentra en el registro de activación del invocado.

- Borrar el registro de activación del invocado se logra restando del apuntador de la pila, el tamaño del registro de activación del invocado:

SP := SP – tamaño(RA del invocado)

Implementación de los Lenguajes Estructurados

- Reinstalar el contexto del invocador como el contexto activo se logra cargando el registro del EP de la liga dinámica del invocado:

EP := M[EP].DL;

Puesto que EP ahora apunta al registro de activación del invocador, podemos usarlo para continuar la ejecución del invocador:

goto M[EP].IP;

Este **goto** es, en realidad equivalente a 'IP:=M[EP].IP', de manera que estos dos pasos regresan el lugar del control (el par EP-IP) al invocador.

Implementación de los Lenguajes Estructurados

- El código completo es el siguiente:

```
SP := SP – tamaño(RA del invocado);
EP := M[EP].DL;
goto M[EP].IP;
```

Puesto que la memoria debe ser referenciada por la liga dinámica y el campo del IP, un regreso de procedimiento requiere dos referencias de memoria.

Implementación de los Lenguajes Estructurados

- Recordemos que Pascal permite que los procedimientos y las funciones se pasen como parámetros a otros procedimientos.

- En clase mostraremos un programa en Pascal que ejemplifica esto.

Implementación de los Lenguajes Estructurados

- En el programa mostrado, tenemos un procedimiento Q que toma como parámetro a otro procedimiento. Podemos ver dos invocaciones a Q: una en la que se pasa el procedimiento P y otra en la que se pasa el procedimiento T. Debemos resolver 2 problemas:

  1. ¿Qué debe pasarse a Q para representar P o T?

  2. ¿Cómo debe implementarse la llamada indirecta 'fp(5)' en el cuerpo de Q?

Implementación de los Lenguajes Estructurados

- Comenzaremos por la segunda pregunta, porque nos ayudará a responder la primera.

- Supongamos por un momento que la invocación 'fp(5)' se implementa como cualquier otra invocación, usando el código que vimos anteriormente. ¿Funcionaría hacerlo así?

- Los primeros 3 pasos (transmitir los parámetros, almacenar la dirección donde continuará la ejecución y establecer la liga dinámica) funcionan sin problema. Sin embargo, el paso siguiente, que es recorrer la cadena estática para llegar al ambiente de definición, no puede efectuarse.

Implementación de los Lenguajes Estructurados

- Esto se debe a que necesitamos saber la distancia estática desde la invocación hasta el ambiente de definición, lo cual requiere que sepamos el nivel de anidamiento estático del ambiente de definición.

- Para un procedimiento normal, esto se puede calcular fácilmente. Sin embargo, en este caso, necesitamos saber el nivel de anidamiento estático del procedimiento que pasamos como parámetro (P o T en nuestro ejemplo).

Implementación de los Lenguajes Estructurados

- Desafortunadamente, esto puede variar en tiempo de ejecución, de una invocación a otra de Q. De hecho, el ambiente de definición ni siquiera está en la cadena estática activa cuando se le invoca desde Q usando 'fp(5)'.

- Por tanto, podemos ver que parte de la información que debe pasarse a Q es el ambiente de definición del procedimiento que se le pasa como parámetro.

Implementación de los Lenguajes Estructurados

- Una posible solución a este problema es representar un procedimiento pasado como parámetro como un registro con dos elementos:

- 1. El campo del **IP** (*instruction pointer*) contiene la dirección de entrada del procedimiento actual.

- 2. El campo del **EP** (*environment pointer*) contiene un apuntador al ambiente de definición del procedimiento que se pasa como parámetro.

- A este tipo de registro se le llama **pareja EP-IP** o cerradura (*closure*).

Implementación de los Lenguajes Estructurados

- Si ‘fp’ representa la ubicación en un registro de activación, de la cerradura pasada para un parámetro procedural, entonces ‘fp.EP’ es la parte del ambiente de esta cerradura.

- La liga estática en el registro de activación del invocado, puede establecerse usando:

M[SP].SL := fp.EP;

Esto resuelve el problema de acceder al ambiente de definición del invocado.

Implementación de los Lenguajes Estructurados

- Consideremos nuevamente el código que vimos anteriormente. Hay otro problema. El penúltimo paso asigna espacio para el registro de activación del invocado. Esto requiere saber el tamaño de este registro de activación, el cual depende del número de variables locales declaradas en el procedimiento correspondiente.

- Una solución es pasar esta información junto con la cerradura. Una solución más simple es hacer que esta instrucción de asignación sea la primera de cada procedimiento. En este punto, el compilador sabe exactamente cuánto espacio se requiere.

Implementación de los Lenguajes Estructurados

- El paso final, al entrar al procedimiento, requiere también acceso a la cerradura, puesto que diferentes parámetros procedurales tienen diferentes puntos de entrada. Por lo tanto, la cerradura especifica la dirección de entrada del procedimiento que se pasa como argumento (IP) y el contexto en el cual debe ejecutarse (EP).

- En clase veremos el código resultante.

Implementación de los Lenguajes Estructurados

- Se requieren cinco referencias a memoria (incluyendo dos para acceder el IP y el EP del procedimiento).

- La salida de un procedimiento pasado como parámetro debe ser la misma que la de un procedimiento normal, puesto que un procedimiento puede ser invocado de manera directa o pasarse como parámetro.

Implementación de los Lenguajes Estructurados

- ¿Cómo se construye la cerradura (par EP-IP) para un procedimiento pasado como parámetro?

- La parte del IP es sólo el punto de entrada al procedimiento, el cual es una constante determinada por el compilador.

- La parte del EP es el ambiente de definición del procedimiento, al cual se accede a través de la cadena estática usando el nivel de anidamiento estático de la tabla de símbolos del procedimiento.

Implementación de los Lenguajes Estructurados

- Por ejemplo, en el código que vimos anteriormente, para construir el par EP-IP en la invocación 'Q(P)', debemos seguir una liga de la cadena estática. El código para construir una cerradura en 'M[SP].PAR[1]' es:

```
M[SP].PAR[1].IP := entrada(P);   construir IP
AP := M[EP].SL;                  obtener ambiente
                                 de definición
M[SP].PAR[1].EP := AP;           construir EP
```

Implementación de los Lenguajes Estructurados

- En general, si *sd* es la distancia estática entre la declaración de un procedimiento P y una invocación que usa a P como su iésimo parámetro, entonces el código para construir el par EP-IP para este parámetro es el siguiente:

	M[SP].PAR[i].IP := entrada(P);	construir IP
	AP := M[EP].SL;	recorrer 1ª. liga
(*sd*-1) ×	AP := M[AP].SL;	recorrer las demás
	M[SP].PAR[i].EP := AP;	construir EP

Nótese que se requieren *sd*+2 referencias de memoria: *sd* para acceder al ambiente de definición y 2 para almacenar el EP y el IP.

Implementación de los Lenguajes Estructurados

- Pascal, Algol y muchos otros lenguajes estructurados modernos, permiten el paso de procedimientos y funciones como parámetros a otros procedimientos. Sin embargo, la mayor parte de ellos no permiten regresar una función o un procedimiento.

- La razón es simple. El uso de una pila de registros de activación no sería adecuado en este caso, pues al salir de un procedimiento se borra su registro de activación. Sin embargo, si queremos regresar un procedimiento, su registro de activación debería poder conservarse, pues podría volver a ser invocado. Más adelante, veremos que los lenguajes funcionales usan un modelo diferente para lograr esto.

Implementación de los Lenguajes Estructurados

- Existe otro constructor que necesita manipular la pila en tiempo de ejecución: el **goto**.
- ¿A qué se debe esto? Los **goto**s locales (o sea, que saltan dentro del bloque donde están definidos) son fáciles de implementar, pues sólo requieren de saltos simples a otra posición de memoria.
- Sin embargo, los **goto**s no locales deben restaurar el ambiente del punto a donde realicen su salto (el cual se define mediante una etiqueta). En clase veremos un ejemplo.

Implementación de los Lenguajes Estructurados

- Veremos en clase cómo luce la pila y los registros justo antes y después del '**goto** 1' que se muestra en el código.
- Podemos resumir estas observaciones diciendo que un **goto** transfiere el lugar de control del **goto** a la etiqueta.
- Tanto el sitio (IP) como el contexto (EP) de ejecución, deben ser alterados.

Implementación de los Lenguajes Estructurados

- Restaurar el contexto en el destino del **goto** requiere los registros del EP y del SP. ¿Cómo se logra esto?

- Obtener el ambiente de definición de la etiqueta consiste simplemente en llegar al ambiente de definición de un procedimiento. La tabla de símbolos para la etiqueta contendrá el nivel de anidamiento estático de su definición, de manera que la distancia estática al ambiente se puede calcular en tiempo de compilación como la diferencia de los niveles de anidamiento estático del uso y de la definición de la etiqueta.

Implementación de los Lenguajes Estructurados

- De tal forma, EP puede establecerse (en tiempo de ejecución) en el contexto de la etiqueta, atravesando *sd* veces la cadena estática, donde *sd* es la distancia estática entre el **goto** y la definición de la etiqueta:

$$sd \times \text{EP} := \text{M}[\text{EP}];$$

where sd = snl(**goto**) – snl(etiqueta)

y 'snl(x)' es el nivel de anidamiento estático de *x*.

Implementación de los Lenguajes Estructurados

- El registro del SP apunta a la siguiente posición disponible de la pila, arriba del registro de activación actual, de manera que se puede restaurar usando:

SP := SP + tamaño(RA de la etiqueta);

El tamaño del registro de activación es conocido por el compilador o puede calcularse de la información disponible en el registro de activación.

Implementación de los Lenguajes Estructurados

- El último paso es transferir a la dirección correspondiente a la etiqueta, la cual es una constante disponible al compilador. Los pasos para un **goto** no local pueden resumirse de la manera siguiente:

$sd \times$ EP := M[EP];
SP := EP + tamaño(RA de la etiqueta);
goto dirección(etiqueta);

Podemos ver que el número de referencias de memoria para efectuar un **goto** no local es *sd*, que es la distancia estática del **goto** a la etiqueta.

Costo de Implementación usando el método de la Cadena Estática

Operación	Referencias de Memoria
Acceso a una variable	$sd + 1$
Invocación a un procedimiento	$sd + 3$
Regreso de un procedimiento	2
Pasar un procedimiento como parámetro	$sd + 2$
Invocar un procedimiento pasado como parámetro	5
goto	sd

Método de los Displays

- El método de la cadena estática puede resultar muy costoso si tenemos que acceder a variables no locales en ambientes profundamente anidados.

- Este problema se origina debido a la organización secuencial de la cadena estática. Cada acceso a una variable local requiere recorrer la cadena estática hasta encontrar su ambiente de definición. Esto podría hacerse más eficientemente si hubiese una forma de llegar *directamente* al ambiente de definición.

Método de los Displays

- Esto puede lograrse usando un arreglo que contenga apuntadores a todos los contextos accesibles. Esto es, si 'D' es el arreglo, entonces 'D[i]' es un apuntador al registro de activación para el ambiente en el nivel de anidamiento estático *i*. A este arreglo se le llama **display.**

- En clase veremos una representación gráfica de un **display** y su relación con una pila de registros de activación como las vistas anteriormente.

Método de los Displays

- Nótese que en este caso, no se requiere registro para el EP, puesto que el compilador sabe el nivel al cual se ejecutará cada sentencia.

- Por lo tanto, si una sentencia se va a ejecutar en el nivel *i*, puede encontrar su nivel de activación usando 'D[i]'.

- ¿Cómo se mejora el acceso a las variables usando un **display**? Recordemos que acceder a una variable involucra 2 pasos: (1) localizar su registro de activación y (2) localizar la variable dentro de este registro.

Método de los Displays

- En vez de tener que recorrer la cadena estática, en este caso, podemos llegar al registro de activación de la variable usando 'D[snl]', donde *snl* es el nivel de anidamiento estático de la declaración de la variable. Para el segundo paso, sumamos el desplazamiento de la variable (o*ffset*) a la dirección base obtenida del paso anterior. O sea, usamos:

fetch M [D[snl] + offset]

Esto requiere 2 referencias de memoria: una para obtener el registro de activación de la variable y otro para obtener su ubicación dentro del mismo.

Método de los Displays

- Consideremos ahora una invocación a un procedimiento definido en un nivel de anidamiento d, desde un nivel de anidamiento u. Si el nombre del procedimiento es visible, entonces debe haber sido declarado al mismo nivel de anidamiento estático desde el cual se le está invocando, o a uno más bajo. En otras palabras, $u \geq d$.

- En clase se mostrará gráficamente como luce la pila antes y después de la invocación.

Método de los Displays

- Nótese que si la llamada es desde el nivel *u*, entonces todas las entradas del **display**, de D[1] a D[u] deben estar en uso (es decir, deben contener apuntadores a registros de activación).

- Además, nótese que si el ambiente de definición del procedimiento está al nivel 2, entonces el procedimiento mismo está al nivel d+1, y el apuntador a su propio registro de activación debe ir en D[d+1]. Esto destruirá el contenido previo de D[d+1], el cual estaba siendo usado si $u > d$ (lo cual será frecuentemente el caso). Por lo tanto, el contenido previo de D[d+1] debe almacenarse en algún lugar.

Método de los Displays

- Para lograr esto, debemos establecer un campo llamado 'EP' en el registro de activación del procedimiento, a fin de almacenar el elemento del **display** . Las partes de un registro de activación para el método del **display** son:

- PAR Parámetros
- IP Dirección de continuación
- EP Elemento del display almacenado
- DL Liga dinámica

- No hay campo para la liga estática (SL) porque el **display** hace esta función.

Método de los Displays

- El código requerido para invocar un procedimiento usando **displays** se mostrará en clase.
- Nótese que en este caso, el número de referencias de memoria requeridas (omitiendo la inicialización de parámetros) es 6.
- Esto suele ser más económico que las *sd*+3 referencias de memoria que requiere el método de la cadena estática.

Método de los Displays

- El regreso de un procedimiento, involucra las operaciones opuestas que la invocación. El IP y el elemento del **display** deben restaurarse desde el registro de activación del invocador, y se debe borrar el registro de activación del invocado.

- El código se mostrará en clase.

- Nótese que restaurar el display eleva el costo: se requieren 5 referencias de memoria para retornar de un procedimiento usando **display**s, en vez de las 2 referencias que requiere el método de la cadena estática.

Comparación entre la Cadena Estática y el uso de Displays

Operación	Cadena Estática	Display
Variable local	1	2
Variable no local	$sd + 1$	2
Invocación a un procedimiento	$sd + 3$	6
Regreso de un procedimiento	2	5