TEMA 2. Ampliación del lenguaje y ampliación de la máquina virtual:

- instrucción compuesta, instrucciones de control y subrutinas,

- definición y construcción de tipos,

- **procedimientos y funciones**

Procedimientos y Funciones

Capítulo 7 de Aho: Entorno de ejecución

1. Introducción (capítulo 7 del Aho):

Diferenciar funciones y procedimientos de macro-expansiones y procesos.

Las macro-expansiones no afectan a la generación de código, son solo un proceso de sustitución de texto en el código fuente.

Activar un proceso implicaría que la máquina-p pudiese replicarse a sí misma, podríamos construir una jerarquía de máquinas-p isomorfa a la jerarquía de procesos.

El problema que vamos a tratar en este tema son las funciones y procedimientos.

Una variable representa un lugar en la memoria de la máquina y el valor guardado en ese lugar.

Un procedimiento (función) representa:

- Un conjunto de instrucciones (el cuerpo del procedimiento).

- Una declaración que puede incluir nuevas variables (y/o parámetros) y procedimientos.

- Además una función incluye un valor de retorno (que puede utilizarse como un factor dentro de una expresión).

Los dos últimos puntos son los objetos(datos) asociados al procedimiento formarán parte del entorno asociado al mismo.

En la arquitectura de nuestra máquina-p no hay nada, de momento, que se corresponda con este concepto de función (procedimiento).

Ampliaremos la máquina virtual con un "paquete de apoyo" a la ejecución. El diseño de este "paquete de apoyo" a la ejecución estará determinado por la semántica de las funciones.

El "paquete de apoyo" a la ejecución debe especificar claramente las tareas asociadas a las siguientes actividades:

- activación (ejecución) de un procedimiento.
- mantenimiento de los procedimientos activos incluyendo diferentes activaciones del mismo procedimiento.
- manipulación de los "objetos de datos" o áreas de datos asignados a cada procedimiento activo.

2. Puntos importantes (siguiendo al Aho)

2.1.- Procedimientos: texto y acciones.

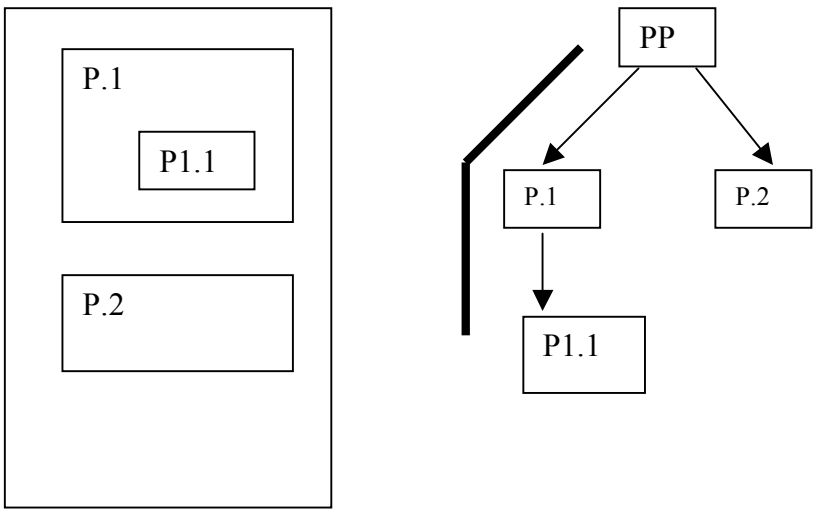
Distinguir entre el **texto** fuente de un programa con procedimientos y las **acciones** realizadas durante la ejecución de este programa.

Para los procedimientos necesitamos un entorno espacio temporal más elaborado que el que habíamos utilizado hasta ahora para las variables y constantes simbólicas.

In programming language semantics, the term "environment" refers to a function that maps a name to a storage location, and the term "state" refers to a function that maps a storage location to the value held there. Using the terms l-value and r-value, an "environment" maps a name to an l-value, and a "state" maps the l-value to a r-value (Aho et al.)

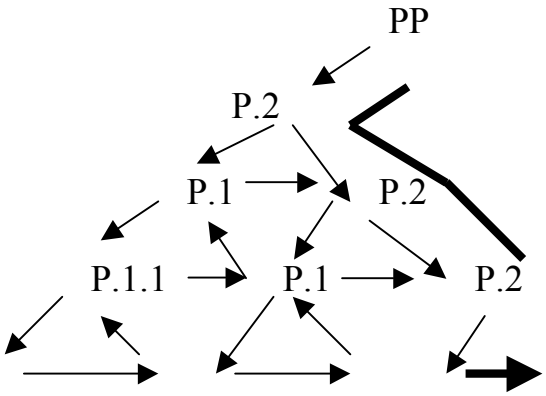
Texto y entorno Léxico: estructura <u>estática</u>

- La gramática independiente de contexto utilizada para generar el texto fuente de un programa da a éste una estructura de árbol (quizá descrito mediante bloques anidados)
- una pila puede representar el nivel de anidamiento de los sub-árboles ó bloques de texto (procedimientos) en un determinado punto del texto.
- esta estructura está relacionada con:
    - la estructura de la t**abla de símbolos** durante su compilación
    - la estructura de la **memoria de datos** durante su ejecución.



Ejecución y entorno Dinámico: estructura temporal de las acciones.

- **Árbol de activación (flujo de control)**. Los árboles de activación pueden ser diferentes en cada ejecución del mismo programa.



- **Pila de control**. Una pila representa (durante la ejecución) el nivel de anidamiento de los procedimientos activos en un instante. Esta estructura de pila está relacionada con la estructura de la memoria de datos, pila de registros de activación, gestionada por el paquete de apoyo a la ejecución.

Al no tener un espacio textual y temporal único surgen dos conceptos importantes:

- **Ámbito** de validez de un nombre

  ¿cómo y cuándo se establece la ligadura de los nombres y cúal es su ámbito de validez?
  - estática: sobre el texto fuente (pila del entorno léxico)
  - dinámica: sobre la memoria de datos (pila de ejecución)

- **Duración de la ligadura** de un nombre,

  ¿cómo y cuándo se establece la ligadura de los nombres? ¿cuánto dura esta relación?

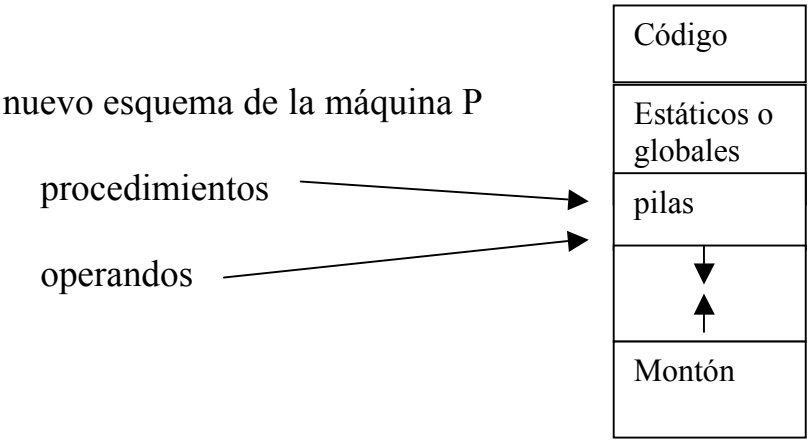
  entorno(ambiente): nombres --> memoria (direcciones)
  estado(almacén): memoria --> valores

- estática : realizada por el compilador al construir, en base al texto, las tablas de símbolos (pila de tablas de símbolos).

- dinámica: realizada por el "paquete de apoyo" al activar un procedimiento. Relacionada con la estructura (**pila**) de los registros de activación (marcos o frames)

Es importante notar que utilizar en los apartados siguientes una pila, en vez de un árbol, para representar los entornos léxico y dinámico supone realizar una elección en cuanto al ámbito y tiempo de vida de los nombres del programa.

2.2.- Organización de la memoria



Pila de registros de activación (marcos): es el área de datos (memoria) dinámicamente asociada con la activación y desactivación de procedimientos.

El nombre de un procedimiento no solo representa un conjunto de instrucciones el área de datos donde se ejecuta.

Una posible organización del registro de activación de un procedimiento es la siguiente:

- resultado de la función
- parámetros
- información de control (estado de la máquina, mantenimiento del entorno de las funciones y del control)
- variables locales
- temporales

Observaciones:
- La disposición de estos elementos dentro del registro de activación puede variar dependiendo del modelo de ejecución que elijamos para nuestra máquina virtual (modelo de enlaces o *display*).

- También depende de la forma en que implementemos el paso de parámetros y la obtención del resultado de la función.

- El orden en que estos elementos están dados en la lista anterior se corresponde con una pila de funciones integrada con la pila de operandos, correspondiendo esta última a los temporales del registro de activación.

discusión: dar un posible esquema de la pila de registros de activación y de la de operandos (temporales)

2.3.- Estrategias para la asignación de memoria

asignación estática (nuestra vieja máquina-p y FORTRAN)
asignación dinámica: pila de registros de activación
asignación dinámica: montón (procesos)

2.4.- Tablas de símbolos

- La estructura básica es una pila (sería posible un árbol si eligiésemos formas más complejas de determinar el ámbito de las variables)

- Construcción: *arrays*, listas enlazadas, tablas *hash*

2.5.- Acceso a los nombres no locales

bloques y ámbito léxico
- ámbitos léxicos definidos exclusivamente por las funciones
- definidos además por otras estructuras sintácticas:
  marcadores específicos, instrucciones de control, etc.

procedimientos no anidados (C por ejemplo)
procedimientos anidados (Pascal por ejemplo)

Mantenimiento del ámbito léxico en diferentes modelos de ejecución: enlaces y *display*

2.6.- Paso de parámetros

por valor
por referencia
por nombre

---

3. Construcción de un entorno de ejecución que admita funciones (procedimientos) anidados

Tenemos que hacer cambios en el compilador y en la máquina-P

Hasta ahora considerábamos que:

- nuestros programas se componían de un único procedimiento o función
- la memoria de datos de la máquina-P se correspondía con el "área de datos", "marco" o "registro de activación" de este único procedimiento.

Durante la generación de código, el compilador, utilizando una función de entorno, asignaba, de forma estática, direcciones de memoria a los nombres (el atributo dirección del símbolo) al construir la tabla de símbolos

Entorno: nombres --> direcciones (memoria)

Durante la ejecución la máquina virtual (función de estado) obtenía los valores asociados a las direcciones

Estado(almacén): memoria --> valores

Si tenemos procedimientos anidados tendremos que modificar:

- el modelo de tabla de símbolos para tener en cuenta el ámbito de validez de la declaración de los nombres durante la generación de código.

- el modelo de ejecución (estructura de la máquina-p) para tener en cuenta el tiempo de vida de estas ligaduras y su ámbito de validez durante la ejecución.

Estos dos cambios están relacionados entre sí. La dirección que se asigne a un nombre (símbolo) al construir la tabla de símbolos tendrá que ser

relativa al bloque de memoria que, en la máquina virtual, se asigne al procedimiento que lo define durante su ejecución.

Cambios que afectan a la construcción de las tablas de símbolos

Tablas de símbolos:

- Independientes.
  - Tablas de símbolos independientes para cada función. Cada función delimita un área (bloque) del texto fuente.
  - Durante la ejecución, suponemos un área de datos (registro de activación) distinta para cada función .
  - En algunos lenguajes se pueden definir además bloques de ámbito de validez de los nombres dentro de las funciones, pero estos casos se tratan de forma estática sin ninguna repercusión en el ámbito de ejecución.

- Organización.
  - La estructura de árbol, que la gramática del lenguaje da al texto de un programa, nos sugiere utilizar una pila.
  - Podemos basarnos en la pila de procedimientos que define cada elemento del árbol del programa fuente para definir los criterios de nuestro lenguaje con respecto al ámbito de validez de los nombres locales y no locales en cada procedimiento.

  Puntos a decidir:
  ¿qué nombres son visibles desde un punto del texto?
  ¿cómo se resuelve la colisión de nombres?

- Información asociada a los nombre de los símbolos (variables, constantes, procedimientos, ..):

  - Información asociada a las variables:
    Direcciones. Las direcciones de las variables son relativas al comienzo del área de datos de la función (cuando solo había una esta podía ser el comienzo del propio área de datos).
    Nivel. Referido a la estructura del texto. Nivel en que ha sido declarada la variable. Necesitamos esta información para determinar los nombres no locales accesibles durante la ejecución de un procedimiento o función.

  - Información asociada a los procedimientos y/o funciones:
    Etiqueta. Dirección referente a un área de código única.

Nivel. Información que permite reconstruir el entorno léxico (la estructura textual de la declaración de procedimientos) durante la ejecución.

Es importante la distinción entre la declaración del nombre de un procedimiento y cuerpo del procedimiento ¿en tablas distintas?

Generación de código: modificaciones en la máquina-P

Ampliación de la máquina-P (nuevas instrucciones y cambios en otras):

Funciones y procedimientos

Estas instrucciones deben construir y mantener los entornos léxico y dinámico en la memoria de datos durante la ejecución.

Necesitamos mantener dos pilas superpuestas en la memoria de datos durante la ejecución:

- la que describe el entorno léxico (estático) del procedimiento activo
- la que describe su entorno dinámico, procedimientos pendientes de finalizar su ejecución.

llamada ; f(np) ; etiq

Llamada: información para ceder control, construir y mantener el entorno léxico y dinámico durante la ejecución.

f(np) = "nivel del procedimiento" sirve para reconstruir el entorno léxico en ejecución

etiq = "dirección dónde comienza el código" sirve para ceder control.

- Salva el *status* de la máquina (p.e. salvar el CP en el registro de activación antes de substituirlo por *etiq*)
- Construye la pila de registros de activación y la pila léxica

incrementar ; <nº de posiciones>

nº posiciones = f(resultado+parámetros+control+variables locales)

- Después de producirse la llamada a un procedimiento, esta instrucción completa la creación del registro de activación.

discusión: simular la ejecución de un programa con un procedimiento recursivo

retorno (nivel)
Retorno: mantiene el modelo de ejecución de las funciones.
La información que necesita la puede tomar (al menos parcialmente) del área de datos del procedimiento que termina:

- restaura el *status* de la máquina (el CP)
- restaura el estado de la memoria de datos: desapila el registro de activación del procedimiento terminado y restaura la pila léxica.

Si es una función que puede retornar un valor, hace posible que el procedimiento llamador pueda recuperar ese valor.

Variables

- Al cambiar el ámbito léxico y la duración, o tiempo de vida, de las ligaduras de las variables debemos modificar las instrucciones que referencian variables.
- Un nuevo argumento (en función del nivel en que la variable ha sido declarada) sirve para determinar el acceso al área de datos que corresponde a la variable (dependiendo de que la variable sea local, o no local)

despila-dir ; f(nv) ; dir
desapila-ind necesita dos posiciones de la pila: f(nv), dir
apila-dir ; f(nv) ; dir
apila-ind necesita dos posiciones de la pila: f(nv), dir

Modelos de ejecución

Modelo de enlaces (inicialmente sin paso de parámetros ni retorno de valor)

1. Registro de activación (puede incluir temporales, o no, en nuestro caso sería la pila de operandos).

   Necesitamos añadir a la máquina-p un registro **C** que apunta a la cima de la pila de registros de activación (podría ser común con la

pila de operandos si incluimos los temporales en el registro de activación).

2. Añadir a la máquina un registro base **B**.

   Es la base para las direcciones relativas del registro de activación. Apunta a la base del registro de activación activo: el que está en la cima de la pila (anteriormente y de forma implícita siempre apuntaba al comienzo de la memoria de datos)

3. Instrucción de LLAMADA.

   El argumento "etiq" se copia en el **CP (contador de programa)** , pero antes el contenido de **CP** se salva en el nuevo registro de activación (como **IS: instrucción siguiente)**. El contenido de **B** (del llamador) se salva en el nuevo registro de activación (como **ED: enlace dinámico**, permite regresar al procedimiento llamador).

   El enlace estático, **EE**, el enlace al padre léxico del procedimiento, también se coloca en el nuevo registro de activación y se obtiene a partir de **B** utilizando tantos niveles de indirección como indique

   f(nv)= na-nd (nivel actual - nivel de definición),

   este valor inicialmente siempre es cero.

   El registro **C** se copia en **B**, es la base del nuevo reg. de activación (las tres posiciones de memoria **IS**, **ED** y **EE** constituyen el estatus o información de control salvado en el registro de activación creado)

   discusión: simular una ejecución que utilice procedimientos anidados, procedimientos hermanos y procedimientos recursivos

4. Instrucción INCREMENTAR **C**.

   Terminada la instrucción de llamada, **B** y **C** apuntan ambos a la base del nuevo registro de activación.

   Para terminar de construir el nuevo registro de activación incrementamos **C** en un valor igual al número de posiciones de memoria necesarias para guardar información de control (una constante que depende de la máquina, igual a 3 en el caso del modelo de enlaces) y el número de posiciones de memoria necesarias para las variables locales (depende del número de

variables locales y su tipo, del número de parámetros y el valor de retorno de la función y sus tipos). Este valor es el argumento de la instrucción.

5. Instrucciones que referencian las variables.

   f(nv)= na-nv, este valor se interpreta partiendo de **B** como el nivel de indirección para conocer la dirección base necesaria para calcular la posición de la memoria de datos a la que hay que acceder.

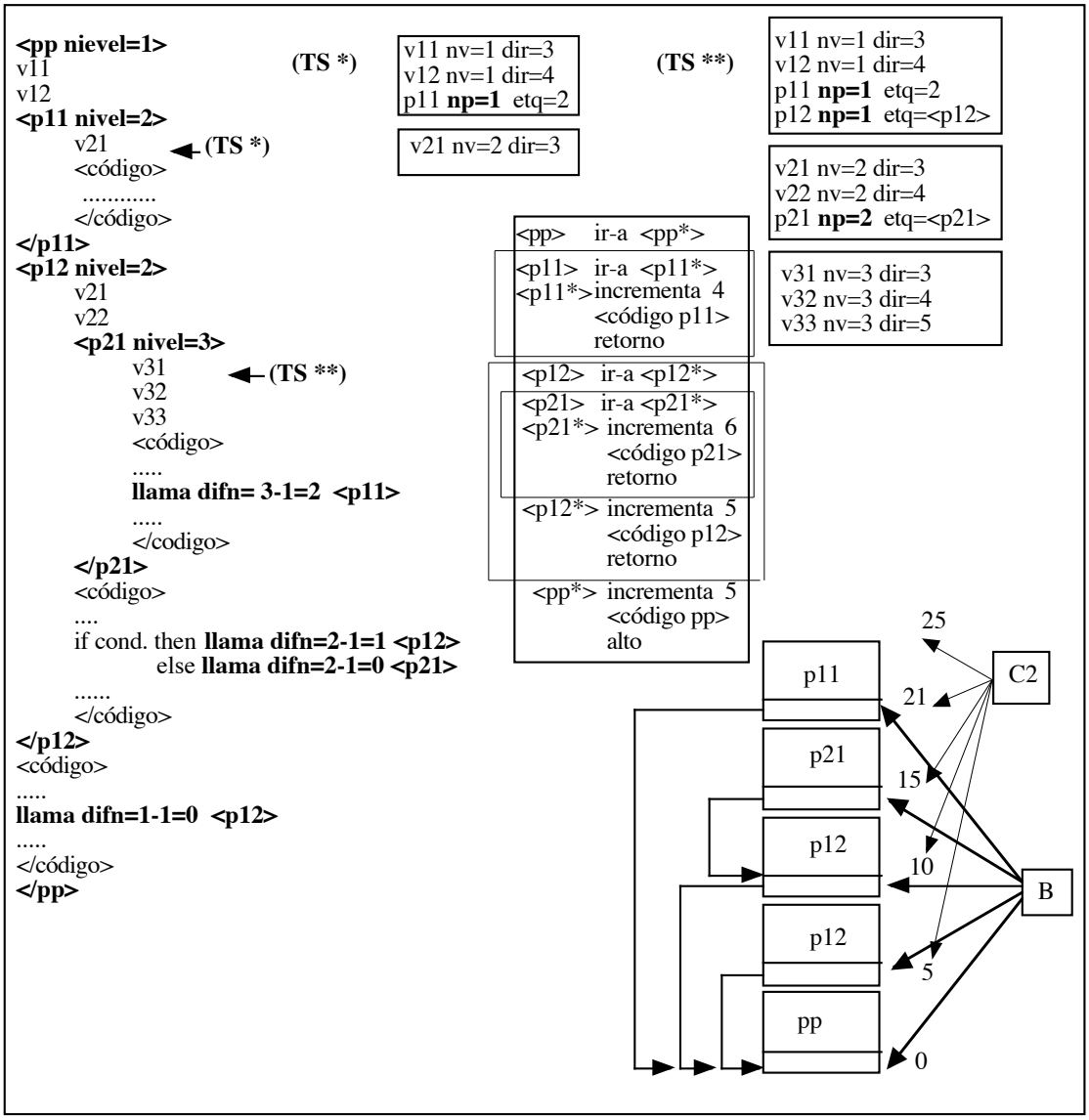
discusión: ejemplos de acceso a variables locales y no locales

6. Instrucción de RETORNO.
   En este modelo no tiene argumentos, restaura el estado de la máquina cuando finaliza una función, utilizando la información de control que hay en el registro de activación.

   - Copia **B** en **C** (desapila el registro de activación del procedimiento que ha finalizado)
   - Copia el **ED** en **B** (restaura la dirección base del procedimiento que realizó la llamada)
   - Copia la **IS** en el **CP** (restaura el control a la instruicción siguiente a la que realizó la llamada).

   Si hubiese valor de retorno se haría accesible este valor al procedimiento llamador.

Si colocamos la pila de operandos en la cima de la pila de registros de activación , el registro cima de ambas pilas se resume en uno solo y podemos considerar los operandos como los temporales de la función activa.



Insrucciones Maquina-P (enlaces)

C1 cima de la pila de operandos (C1 es la primera posición vacía),
C2 cima de la pila de área de datos.

Inicialmente suponemos que no hay paso de parámetros ni valor de retorno, y que utilizamos una pila de operandos independiente de la pila de registros de activación.

Conceptualmente puede resultar más claro tener dos pilas diferentes para funciones y operandos, aunque en la práctica su implementación puede resultar más compleja.

B base de direcciones relativas en la memoria de datos (base del registro de activación activo)
na: nivel activo, en el se está ejecutando la instrucción
nd: nivel declarativo, en el que se declaró el nombre

IMPORTANTE para una variable local na = nd pero para un procedimiento que se llama recursivamente na = nd + 1 (el nombre del procedimiento se considera que pertenece al entorno del padre léxico)

dif.np = na – nd

llamar; dif.np; etiq

```
Datos[C2+2]=CP          (IS)
Datos[C2+1]=B           (ED)
base=B
while dif.np > 0 do
        base=Datos[base]
        dif.np=dif.np-1
Datos[C2]=base          (EE)
B=C2                    base del nuevo reg. de activación
CP=etiq
```

Construcción de la pila léxica superpuesta a la pila de activación. Inicialmente el programa principal llama a uno de sus procedimientos, lo que implica una diferencia de niveles cero. El enlace estático EE coincide con el registro B y con el enlace dinámico ED. El padre léxico y el procedimiento llamador son el mismo.

Este procedimiento llamado en primer lugar puede llamarse a si mismo ó llamar a un procedimiento definido antes que él en el programa principal. En ambos casos la diferencia de niveles es 1, el enlace EE no será el procedimiento llamador si no el padre léxico del procedimiento llamador, que es el padre léxico del procedimiento llamado.

Si el procedimiento llamado llama a uno de los procedimientos que el mismo define la situación es igual que cuando la llamada se hace desde el programa principal.
Si desde un nivel 2 (procedimiento declarado en nivel 1) se llama un procedimiento declarado anteriormente, el padre léxico del llamado será el abuelo léxico del llamador.

La diferencia de niveles sirve par localizar en la ascendencia léxica del llamador: el padre léxico del llamado. El llamador solo puede llamar a procedimientos cuyo padre léxico esté en la ascendencia léxica del llamador.

incrementarC2 <3+mem.var.locales>

```
C2=C2+ <arg>
```

Como antes, suponemos que no hay parámetros ni valor de retorno ó si los hay los contamos como variables locales.

retorno

```
C2=B
B=Datos[C2+1]
CP=Datos[C2+2]
```

dif.nv = na – nd

apilar-dir; dif.nv; dir

```
base=B
while dif.nv > 0 do
        base=Datos[base]
        dif.nv=dif.nv-1
P[C1]=Datos[base+dir]
C1=C1+1
```

apilar-ind (P[C1-1]=dif.nv; P[C1-2]=dir)

```
base=B
while P[C1-1] > 0 do
        base=Datos[base]
        P[C1-1]=P[C1-1]-1
P[C1-2]=Datos[base+P[C1-2]]
C1=C1-1  (los dos elementos de la cima de la pila se sustituyen por uno)
```

desapilar-dir; dif.nv; dir

```
base=B
while dif.nv > 0 do
        base=Datos[base]
        dif.nv=dif.nv-1
Datos[base+dir]=P[C1-1]
C1=C1-1
```

desapilar-ind (P[C1-2]=dif.nv; P[C1-3]=dir)

```
base=B
while P[C1-2] > 0 do
        base=Datos[base]
        P[C1-2]=P[C1-2]-1
Datos[base+P[C1-3]]=P[C1-1]
C1=C1-3  (elimino los tres elementos de la pila)
```

Discusión ¿cómo se tratan los punteros? dif.nv=na (nd=0)

Modelo del *display (Dijkstra)*

Como en el caso anterior suponemos que inicialmente no hay parámetros ni valor de retorno.

1. determinar en compilación la profundidad de anidamiento, *n*, de un programa concreto y pasar esta información al entorno de ejecución.

2. construir un *display* = *array* con tantos registros como niveles de anidamiento (se puede entender que en este modelo no hay un único registro base). En la práctica podemos colocar el valor *n* en la primera posición de la memoria de datos.

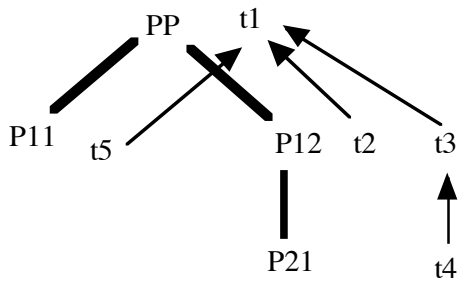
3. LLAMAR. salva el CP en el nuevo registro de activación (será la instrucción siguiente a realizar cuando termine el procedimiento llamado) y hace CP= etiqueta. Salva *display*[np] (el antiguo valor del *display* correspondiente al nivel del procedimiento llamado) en el nuevo registro de activación. Modifica *display*[np] (será el antiguo valor de la cima de la pila de registros de activación) para que apunte a la base del registro de activación del procedimiento llamado.

4. INCREMENTAR, en este modelo solo es necesario reservar dos posiciones de memoria en el registro de activación creado para información de control.

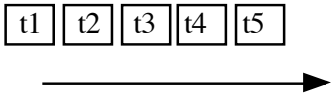
5. acceso a variables. f(nv) =nv. El acceso al área de datos apropiada se hace utilizando el registro del *display* correspondiente al nivel de definición de la variable *display*[nv]+dir

6. RETORNO *np* . Restaura el contador de programa. Restaura el *display* con su antiguo valor (correspondiente al nivel del procedimiento llamado que es el argumento de esta instrucción).

Representamos sobre la estructura estática del programa las instancias de los procedimientos activos en cada instante



La pila de activación sería



atención a como se ajustan los niveles nv np

```
<pp nivel=0>
v11
v12
<p11 nivel=1>
      v21          ← (TS *)
      <código>
       .............
      </código>
</p11>
<p12 nivel=1>
      v21
      v22
      <p21 nivel=2>
            v31
            v32    ← (TS **)
            v33
            <código>
            .....
            llama n=1  <p11>
            .....
            </codigo>
      </p21>
      <código>
      ....
      if condición then llama n=1 <p12>
                  else llama n=2  <p21>
      ......
      </código>
</p12>
<código>
.....
llama n=1  <p12>
.....
</código>
</pp>
```

(TS *)

v11 nv=0 dir=2
v12 nv=0 dir=3
p11 **np=1** etq=2

v21 nv=1 dir=2

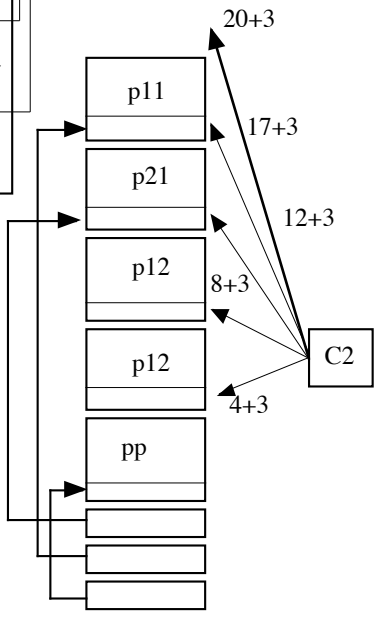
(TS **)

v11 nv=0 dir=2
v12 nv=0 dir=3
p11 **np=1** etq=2
p12 **np=1** etq=<p12>

v21 nv=1 dir=2
v22 nv=1 dir3
p21 **np=2** etq=<p21>

v31 nv=2 dir=2
v32 nv=2 dir=3
v33 nv=2 dir=4

```
<pp>     ir-a  <pp>
<p11>    ir-a  <p11*>
<p11*>   incrementa  3
         <código p11>
         retorno  1
<p12>    ir-a <p12*>
<p21>    ir-a <p21*>
<p21*>   incrementa  5
         <código p21>
         retorno 2
<p12*>   incrementa 4
         <código p12>
         retorno 1
<pp*>    incrementa  4
         <código pp>
         alto
```



Display
Todos los registros de activación de un mismo nivel se referencian mediante un mismo registro. En un momento dado de la ejecución, este registro apunta al registro activo de ese nivel.

El compilador obtiene como un atributo de cada programa concreto el número máximo de niveles de anidamiento de los procedimientos, "n", y para ejecutar el programa pasa este valor junto con el código del programa a la máquina.

En un implementación posible, las "n" posiciones iniciales de la memoria de datos, desde Datos[0] hasta Datos[n-1], son el *DISPLAY*, el conjunto de registros que utilizaremos como base para sumar a las direcciones relativas de las variables y así poder acceder a la posición que realmente ocupan las variables en la memoria de datos. El valor Datos[0]= n es la dirección base para las variables globales, el resto de las posiciones del *display* podemos inicializarlas a - 1.

Leyendo el *DISPLAY* de 0, -->, n tenemos la pila léxica en ese instante de la ejecución.

El registro de activación de los procedimientos solo necesita dos posiciones para la información de control, una para la instrucción siguiente y otra para salvar el contenido de la posición del *display* que se va a re-escribir como consecuencia de la llamada.

IMPORTANTE: En este modelo, el nivel de un procedimiento, np, coincide con el del cuerpo (el mismo que sus variables locales)

na: nivel en se está ejecutando la instrucción
nd: nivel en que se declaró el nombre

np = nd

llamar; np; etiq

Datos[C2+1]=CP	salvar IS en reg de activación
Datos[C2]= Datos[np]	salvar dir base del llamador
Datos[np]= C2	nueva dir base (del llamado)
CP=etiq	

retorno np

CP=Datos[Datos[np]+1]	
C2= Datos[np]	
Datos[np]=Datos[Datos[np]]	restaurar el valor de la dir base anterior

incrementarC2 <2+mem.var.locales>

C2=C2+<arg>

nv = nd

apilar-dir; nv; dir

P[C1]=Datos[Datos[nv]+dir]
C1=C1+1

apilar-ind (P[C1-1]=nv; P[C1-2]=dir)

P[C1-2]=Datos[Datos[P[C1-1]]+P[C1-2]]
C1=C1-1

(los dos elementos de la cima de la pila de operandos se sustituyen por uno)

desapilar-dir; nv; dir

Datos[Datos[nv]+dir]=P[C1-1]
C1=C1-1

desapilar-ind (P[C1-2]=nv; P[C1-3]=dir)

Datos[Datos[P[C1-2]]+P[C1-3]]=P[C1-1]
C1=C1-3 (elimino los tres elementos de la pila de operandos)
