Módulo I

Tipos de datos, estructuras de datos y tipos de datos abstractos.

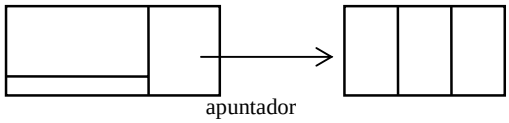
Aunque los conceptos de tipo de datos (TD), estructura de datos (ED) y tipo de datos abstracto (TDA) se han visto en los cursos introductorios de programación, comenzaremos este módulo puntualizando sus características. Es importante destacar que si bien son tres conceptos que tienen que ver con la representación de los datos de un problema, y a veces se los confunde, tienen significados distintos.

En un lenguaje de programación, el **tipo de datos** de una variable[1] está determinado por el conjunto de valores que dicha variable puede tomar y el conjunto de operaciones que se pueden realizar con variables del mencionado tipo (como argumentos y/o como resultado). Por ejemplo en Pascal el tipo boolean tiene dos valores, éste se refiere al conjunto {false, true}, es decir que una variable de este tipo podrá tomar sólo uno de esos valores; con respecto a las operaciones cuenta con la negación, la conjunción y la disyunción cuya representación es: not, + y * respectivamente. En cambio el tipo array [1..3] of boolean cuenta con ocho valores, estando determinado por el conjunto {(false,false,false) , (false,false,true) , (false,true,false) , (false,true,true) , (true,false,false) , (true,false,true) , (true,true,false), (true,true,true)}, donde cada una de las ternas representa uno de los valores que puede tomar una variable declarada del mencionado tipo.

Una **estructura de datos** es una colección de variables (del mismo tipo o no), organizadas de alguna manera determinada. Se considera a la célula como la unidad básica de una estructura de datos.

Además de la capacidad propia de un lenguaje de programación para agrupar las células de una estructura de datos (por ejemplo arreglos, registros, etc.), existe la posibilidad de crear estructuras relacionando o enlazando celdas usando apuntadores.

Un apuntador es un valor que direcciona una determinada célula, es decir que permite que se pueda acceder a ella. Cuando se representa gráficamente una estructura de datos, si un campo de una célula X es un apuntador a otra celda Y, dicho apuntador se representa gráficamente por medio de una flecha que parte del campo correspondiente de X hacia la célula Y.



célula X célula Y

Un apuntador puede implementarse como un cursor (índice de un arreglo) cuando las células son componentes de un arreglo, o como un puntero, si es que el lenguaje de programación cuenta con dicha facilidad. Veremos esto más adelante en detalle.

[1] Puede ser también una constante o el resultado que devuelve una función

Un **tipo de datos abstracto** queda determinado por el modelo matemático que lo sustenta y por un conjunto de operaciones que se definen sobre el mencionado modelo.

En relación con un TDA se puede hablar de definirlo o de implementarlo. Aunque frecuentemente se confunden ambas expresiones, estas tienen significados perfectamente diferenciados.

Definir un TDA es dar el modelo y el conjunto de operaciones correspondientes, expresando con claridad y sin ambigüedad las características de cada una de ellas. Por ejemplo, podría definirse el TDA Número Complejo de la siguiente manera:

- Modelo: par ordenado de números reales.
- Conjunto de operaciones: suma, diferencia, producto, módulo, argumento, leer_número, imprimir_número
- Descripción precisa de cada una de las operaciones

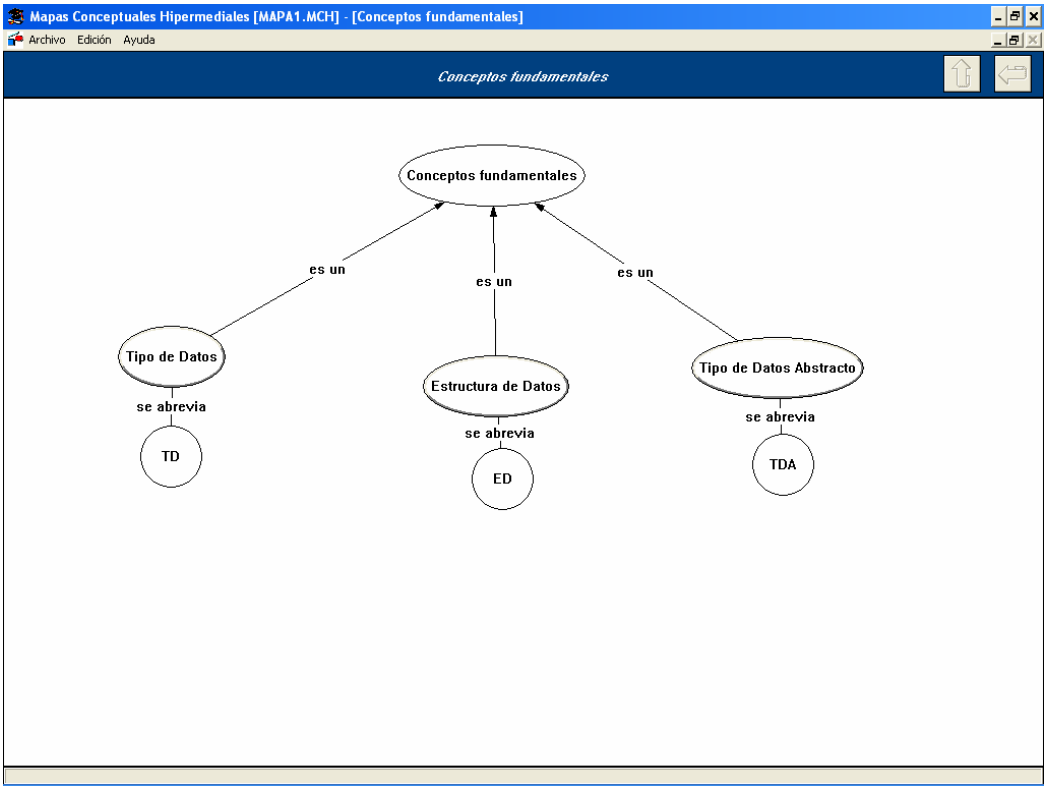
La definición de un TDA debe ser clara y precisa, ya que por un lado está el implementador del TDA quien tomará dicha descripción como base y la seguirá fielmente, y por otro lado está el usuario del TDA que lo usará teniendo en cuenta también lo que indica la definición. Obviamente, para que luego el programa funcione correctamente, ambas cosas deben concordar.

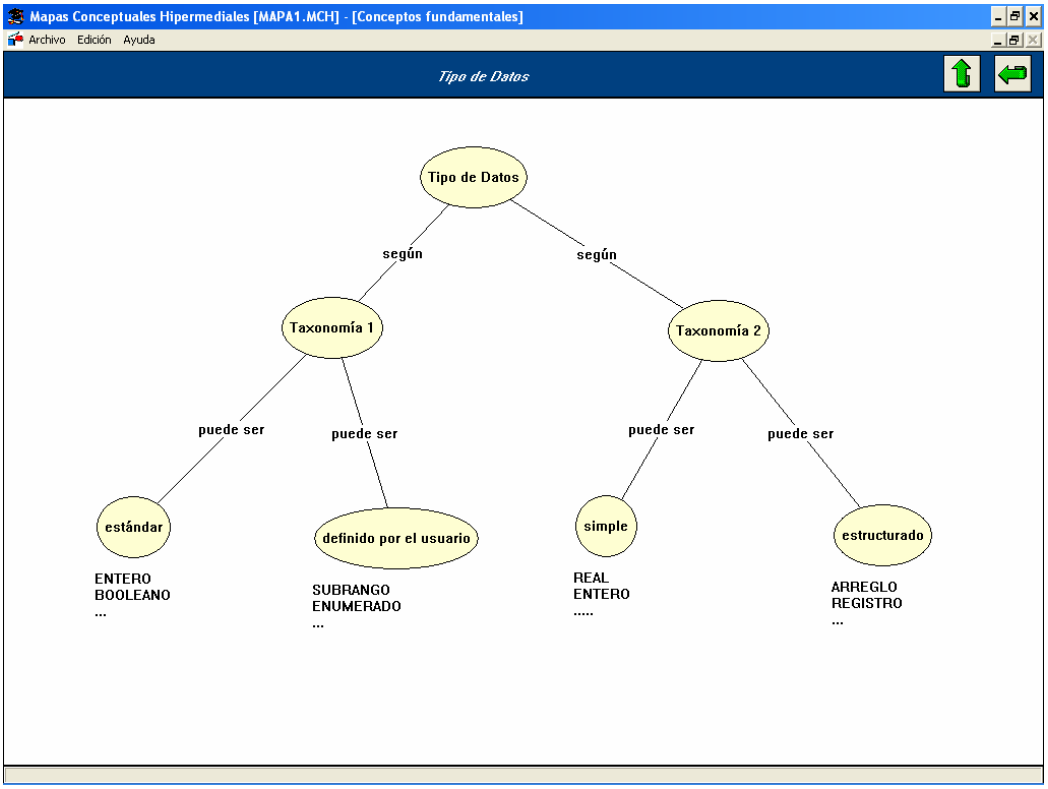
Para **implementar un TDA** se debe encontrar una estructura de datos adecuada (o un TDA estructural) para representar el modelo subyacente del TDA, y se deben escribir los procedimientos (o funciones) que cumplirán al ejecutarse con las tareas propuestas por las operaciones.

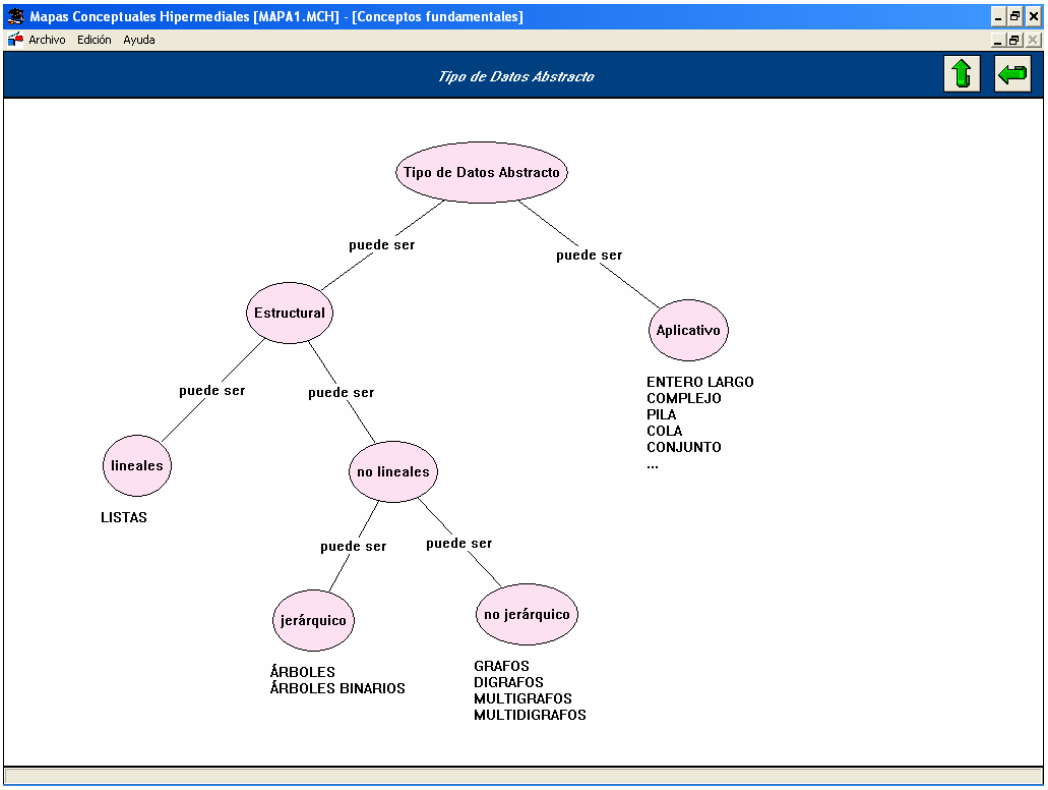
Las definiciones de los TDA se hacen cuando se diseñan los algoritmos, mientras que cuando se implementan los algoritmos en un determinado lenguaje de programación es cuando también se implementan los TDA que se han definido previamente, si es que no se cuenta ya con su implementación. Cabe señalar que si el lenguaje de programación con el que se implementa el algoritmo cuenta entre sus tipos estándar con uno que concuerda perfectamente con la definición realizada para un determinado TDA, se podrá usar dicho tipo sin necesidad de realizar una implementación especial para el TDA en cuestión.

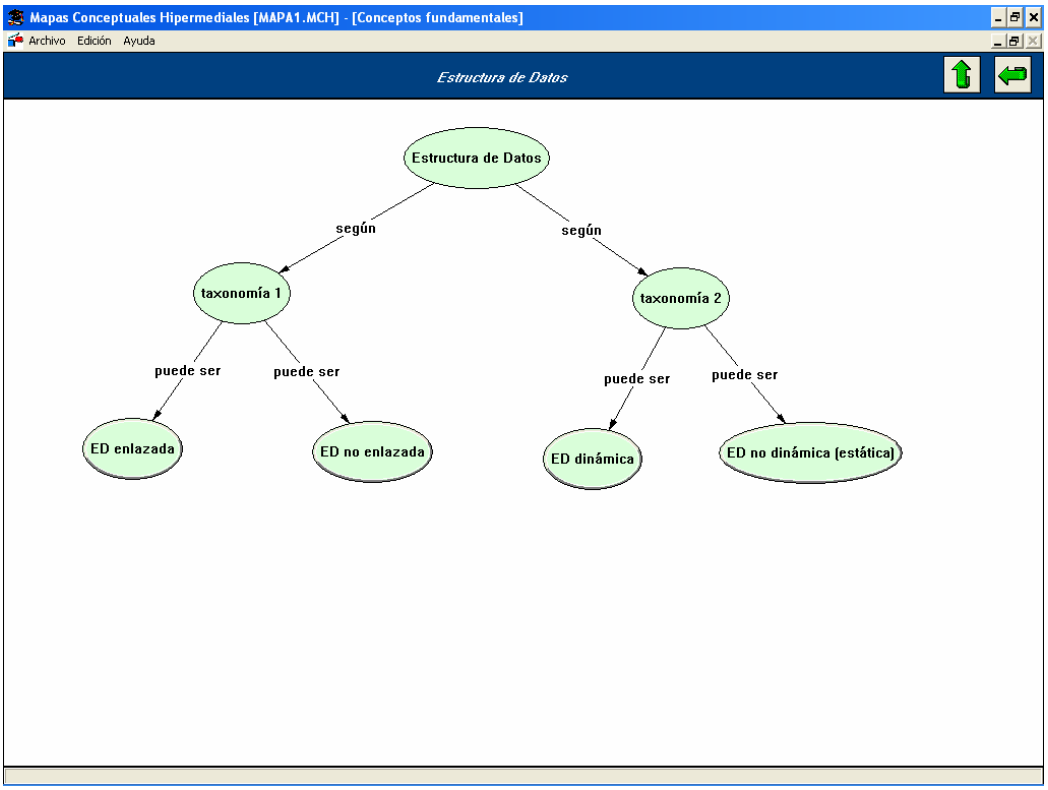
Se presenta a continuación una red conceptual[2] que muestra con claridad la clasificación de los tres conceptos enunciados (TD, ED y TDA) según se trabajará en el curso a lo largo de todo el cuatrimestre.

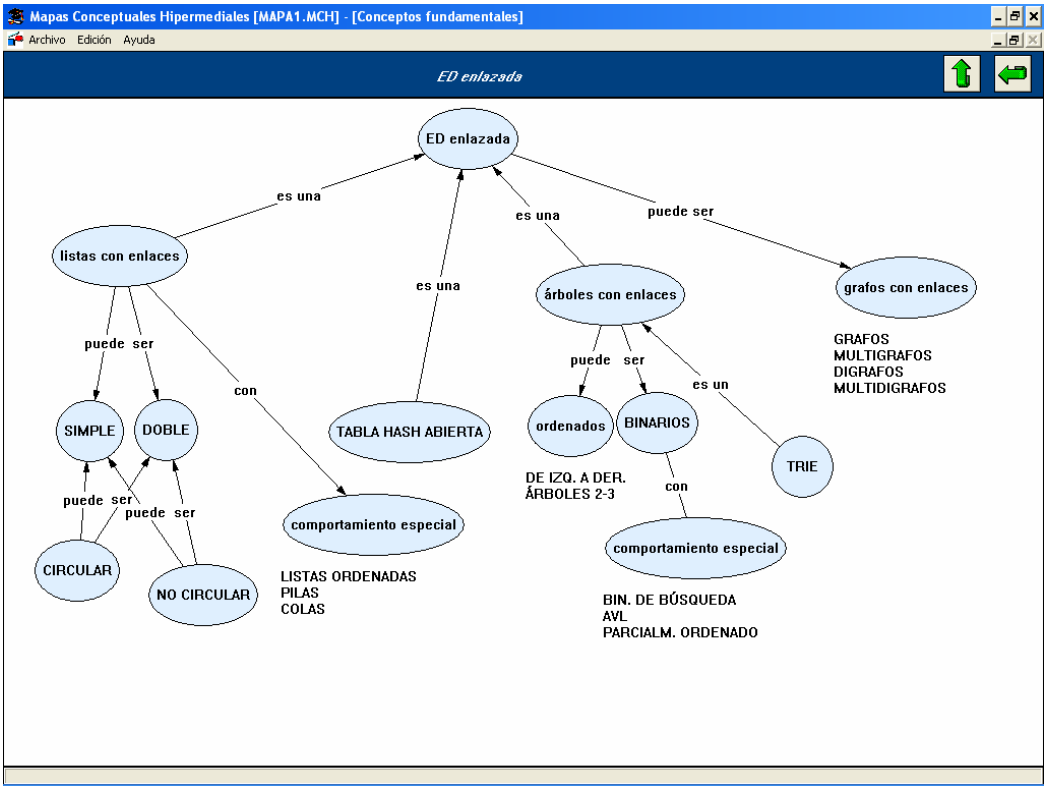
[2] La red conceptual se muestra a través de un Mapa Conceptual Hipermedial (MCH)

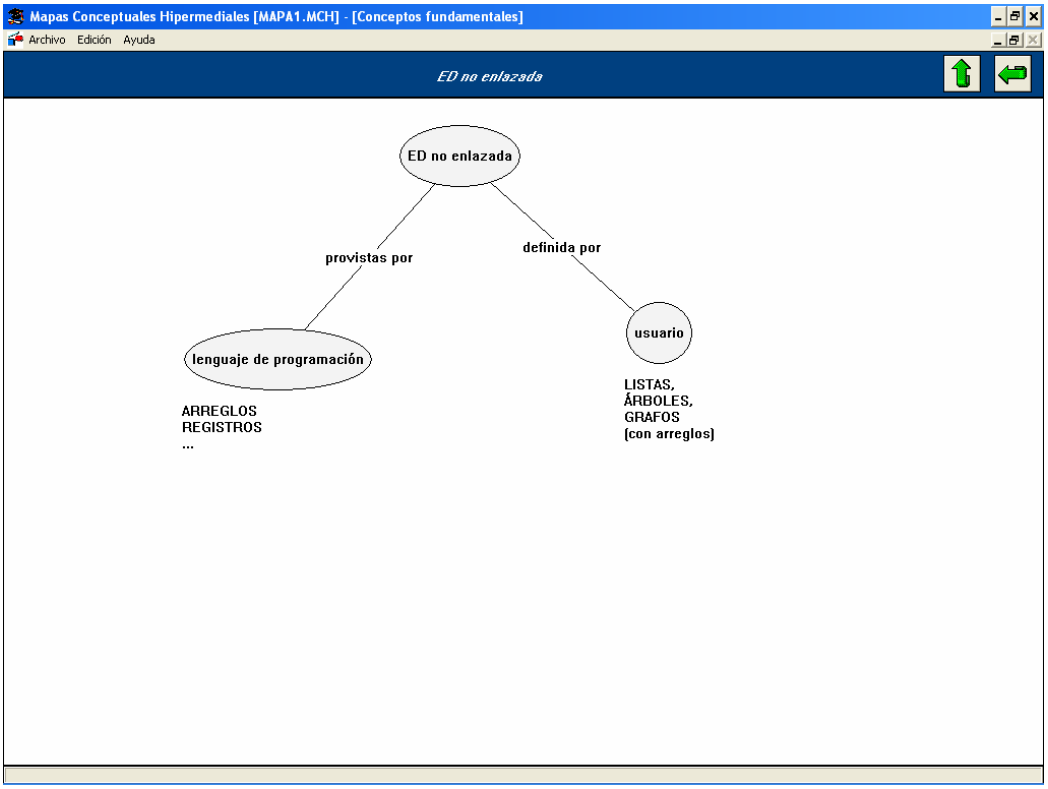

Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
Conceptos fundamentales
Conceptos fundamentales
es un
es un
es un
Tipo de Datos
Estructura de Datos
Tipo de Datos Abstracto
se abrevia
se abrevia
se abrevia
TD
ED
TDA

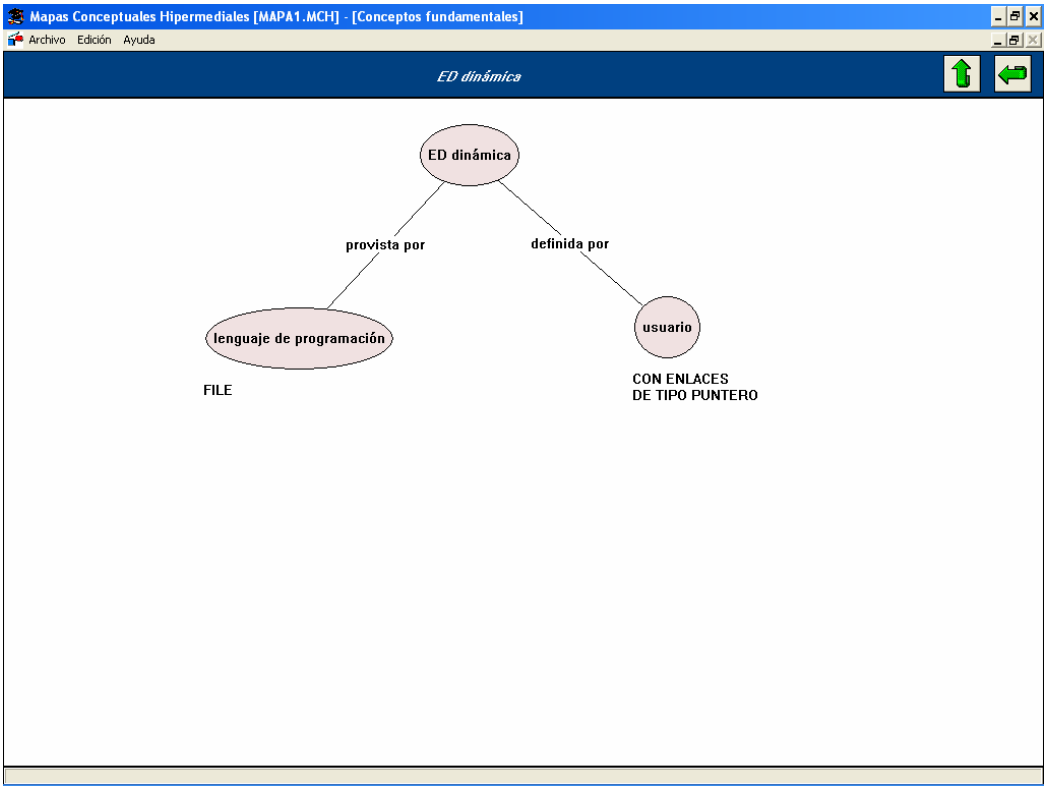

Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
Tipo de Datos
Tipo de Datos
según
según
Taxonomía 1
Taxonomía 2
puede ser
puede ser
puede ser
puede ser
estándar
definido por el usuario
simple
estructurado
ENTERO
BOOLEANO
...
SUBRANGO
ENUMERADO
...
REAL
ENTERO
.....
ARREGLO
REGISTRO
...

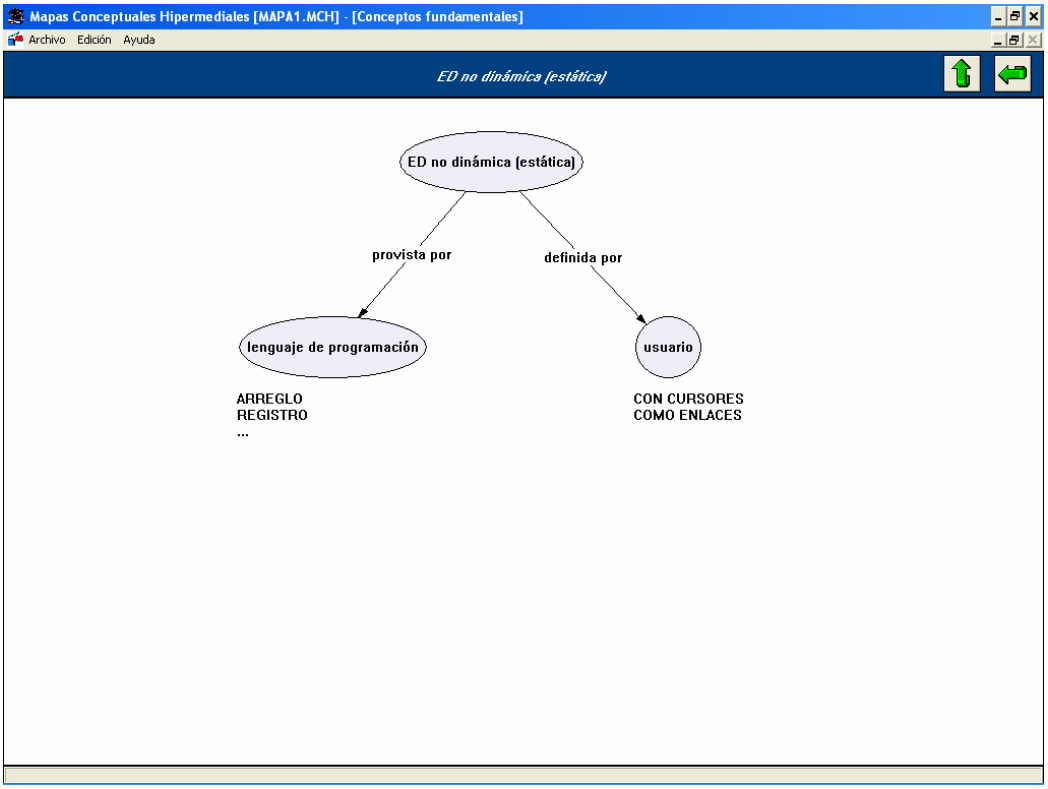

Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
Tipo de Datos Abstracto
Tipo de Datos Abstracto
puede ser
puede ser
Estructural
Aplicativo
puede ser
puede ser
ENTERO LARGO
COMPLEJO
PILA
COLA
CONJUNTO
...
lineales
no lineales
LISTAS
puede ser
puede ser
jerárquico
no jerárquico
ÁRBOLES
ÁRBOLES BINARIOS
GRAFOS
DIGRAFOS
MULTIGRAFOS
MULTIDIGRAFOS


Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
Estructura de Datos
Estructura de Datos
según
según
taxonomía 1
taxonomía 2
puede ser
puede ser
puede ser
puede ser
ED enlazada
ED no enlazada
ED dinámica
ED no dinámica (estática)


Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
ED enlazada
ED enlazada
es una
es una
es una
puede ser
listas con enlaces
árboles con enlaces
grafos con enlaces
GRAFOS
MULTIGRAFOS
DIGRAFOS
MULTIDIGRAFOS
puede ser
puede ser
es un
con
SIMPLE
DOBLE
TABLA HASH ABIERTA
ordenados
BINARIOS
TRIE
DE IZQ. A DER.
ÁRBOLES 2-3
puede ser
puede ser
con
CIRCULAR
NO CIRCULAR
comportamiento especial
comportamiento especial
LISTAS ORDENADAS
PILAS
COLAS
BIN. DE BÚSQUEDA
AVL
PARCIALM. ORDENADO


Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
ED no enlazada
ED no enlazada
provistas por
definida por
lenguaje de programación
usuario
ARREGLOS
REGISTROS
...
LISTAS,
ÁRBOLES,
GRAFOS
(con arreglos)


Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
ED dinámica
ED dinámica
provista por
definida por
lenguaje de programación
usuario
FILE
CON ENLACES
DE TIPO PUNTERO


Mapas Conceptuales Hipermediales [MAPA1.MCH] - [Conceptos fundamentales]
Archivo Edición Ayuda
ED no dinámica (estática)
ED no dinámica (estática)
provista por
definida por
lenguaje de programación
usuario
ARREGLO
REGISTRO
...
CON CURSORES
COMO ENLACES

Estructuras dinámicas

Existen estructuras de datos, como los arreglos o los registros de Pascal, para los cuales se conoce en tiempo de compilación el espacio que ocuparán durante la ejecución del procedimiento (o programa) donde se encuentran declaradas. Además de conocerse con anticipación la cantidad de memoria que ocuparán durante la ejecución, al comenzarse a ejecutar el procedimiento donde se encuentran declaradas, se reserva el espacio de memoria correspondiente y se mantiene reservado mientras dura la ejecución de dicho procedimiento, independientemente de la necesidad de su uso.

En cambio, se ha visto otra estructura, el file de Pascal, donde no es necesario declarar la cantidad de componentes que tendrá. En este caso, obviamente, no se conoce en tiempo de compilación el espacio que ocupará la mencionada estructura durante la ejecución, y además al comenzarse a ejecutar el procedimiento donde la estructura está declarada no es posible hacer una reserva de espacio para todas sus componentes, sino que se va haciendo la reserva de espacio a medida que se necesita en el transcurso de la ejecución.

Se dice que una estructura es dinámica cuando el espacio de almacenamiento en memoria se va obteniendo durante la ejecución, a medida que se lo necesita. Por lo tanto la cantidad máxima de memoria que ocupará la estructura de datos durante la ejecución del procedimiento donde está declarada, no se conoce hasta el momento en que finaliza dicha ejecución.

En el desarrollo de un gran número de programas es conveniente trabajar con estructuras dinámicas. Existe una gran variedad de este tipo de estructuras, tales como listas, árboles, grafos, que veremos en detalle a lo largo del curso.

En contraposición con el nombre de estructuras dinámicas, algunos autores llaman estructuras estáticas a aquellas para las cuales es posible conocer en tiempo de compilación el espacio que ocuparán durante la ejecución.

Lenguajes de programación como Pascal o Modula-2 (que veremos en detalle más adelante) cuentan con herramientas poderosas que son los punteros y la posibilidad de crear variables dinámicas. Estos elementos permiten crear con comodidad estructuras dinámicas. No todos los lenguajes de programación cuentan con esta posibilidad.

Punteros y variables dinámicas.

El conjunto de valores de un tipo de datos puntero es un conjunto de direcciones de memoria. Es decir que una variable de tipo puntero puede tomar como valor únicamente una de esas direcciones de memoria.

Si bien los punteros no son una exclusividad del lenguaje Pascal, las descripciones y ejemplificaciones se harán para este lenguaje en particular.

Muchos lenguajes, incluyendo Pascal, permiten crear y destruir variables durante el transcurso de la ejecución de un programa, se trata de las variables dinámicas.

Creación de una variable dinámica

Una variable dinámica no es declarada en la sección de declaración de variables, como ocurre con las variables que no son dinámicas (variables estáticas). Una variable dinámica es referenciada por medio de una variable de tipo puntero, es decir que en la variable de tipo puntero está la dirección de memoria de la variable dinámica.

Cuando se desea crear una variable dinámica, se hace por medio de un procedimiento estándar que permite que el puntero p tome como valor la dirección de la variable dinámica que queda apuntada por p. A partir del momento que es creada la variable dinámica que está en la dirección que contiene p, se puede acceder a ella por medio del identificador p^ . En Pascal ese procedimiento es NEW(p).

Declaración de variables de tipo puntero

Cuando se declara una variable de tipo puntero p, hay que indicar expresamente de qué tipo será la variable a la que apunta, es decir la variable dinámica correspondiente.

Declaración: var
p: ^T , donde T es un identificador conocido en ese momento, predefinido o declarado previamente y que no es de tipo puntero.

Esta declaración se lee ‘p es una variable de tipo apuntador a una variable de tipo T’.

Se le puede dar un identificador al tipo, declarando:

```
type
    TipoPuntero = ^ T ;
var
    p: TipoPuntero;
```

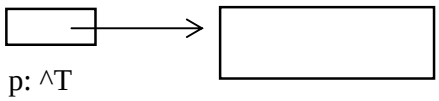
En este caso si T no es predefinido, puede ser declarado antes o después de la declaración de TipoPuntero, indistintamente.[3]

Hay que destacar que la mera declaración de p no le otorga a esta variable ningún valor, por lo tanto con dicha declaración no se crea la variable dinámica p^.

Una vez que se ejecuta NEW(p) , p toma un valor y cobra vigencia la variable dinámica p^ (es decir que se puede acceder a ella).

[3] Es el único caso en el lenguaje Pascal donde se puede hacer referencia a un identificador que puede estar declarado posteriormente.

Gráficamente representaremos este hecho de la siguiente manera:



p: ^T

p^ : T

La ejecución del procedimiento NEW(p) hace que se reserve el espacio de memoria suficiente para una variable de tipo T. Como la variable es dinámica ese espacio se reserva en un área especial de la memoria conocida con el nombre de heap.

A partir del momento que es creada una variable dinámica, y mientras no se la inhabilite expresamente (ya veremos cómo), puede accederse a ella a través de su identificador p^, y podemos operar con ella de la misma forma como se opera con cualquier otra variable del tipo T.

Por ejemplo, si el tipo T es un registro donde uno de sus campos es edad, es válido referirse al campo p^. edad de la variable dinámica p^; o si el tipo T es un array[1..5] of integer, se puede hacer referencia a la segunda componente de la variable dinámica p^, poniendo p^[2].

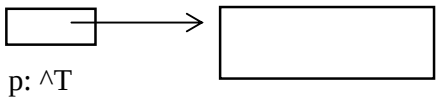
Eliminación de una variable dinámica

Cuando se trabaja con estructuras de datos dinámicas, muy probablemente éstas cambian de tamaño en el transcurso de la ejecución. Teniendo en cuenta que el área de memoria que se reserva en la ejecución del programa para alojar variables dinámicas (el heap) es finita, se debe adoptar como política la de destruir toda variable dinámica en el momento en que no se la use más.

Para que una variable dinámica p^ que ya ha sido creada quede inhabilitada, se usa el procedimiento estándar DISPOSE (p). Este procedimiento deja a la variable puntero p con un valor indeterminado, y al área de memoria que estaba ocupando la variable dinámica p^ disponible para ser usada por otra variable dinámica que se cree posteriormente.

Gráficamente:

Si p^ ya ha sido creada, se tiene:



p: ^T

p^ : T

Si se ejecuta entonces DISPOSE(p), pasa a quedar:

p: ^T

Con el símbolo **?** se está indicando un valor indefinido.

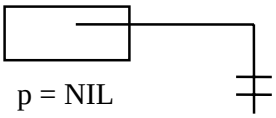
Observar que como el valor de p es indefinido, ésta no tiene más la dirección de la variable dinámica p^, y por lo tanto no se puede acceder más a ella pues no existe más.

Una constante especial

Si p es de tipo puntero, cualquiera sea el tipo de la variable dinámica a la que apunta, puede tomar un valor constante especial: es un valor nulo que se identifica con NIL. Si p tiene valor NIL, significa que no apunta a ninguna variable, es decir que no existe en ese caso p^.

El valor NIL puede ser otorgado a p mediante una sentencia de asignación p := NIL

Gráficamente se representa así:



Observemos que si ha sido creada una variable dinámica p^ y se le asigna luego a p el valor NIL, no podrá accederse a ella, pues p ya ha perdido la dirección de p^.

Comparaciones entre variables de tipo puntero

Dos variables de tipo puntero p y q admiten las comparaciones p=q y p<>q .

Asignaciones de punteros

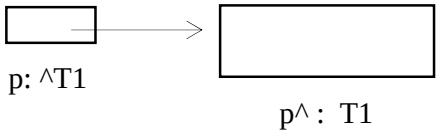
Si existen las variables de tipo puntero p: ^T1 y q: ^T2 , donde T1 y T2 son tipos compatibles, son válidas las asignaciones:

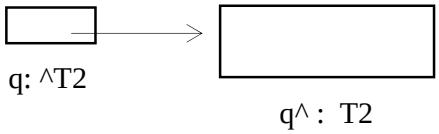
p := q y q := p

Al ejecutarse p := q , tanto p como q quedarán con el mismo valor (el que previamente tenía q)

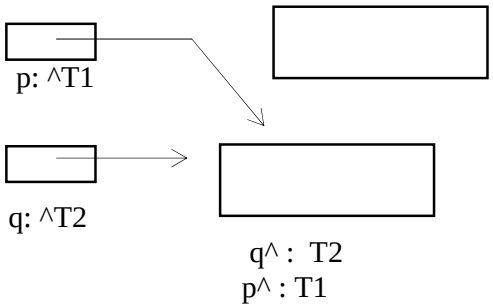
Gráficamente:

a) Si se tenía :





Luego de p := q quedará:

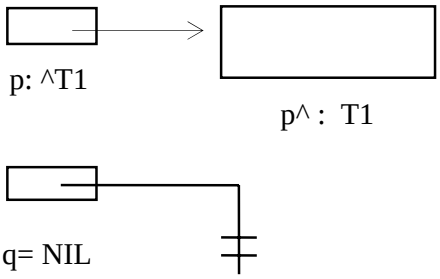


Observemos que:

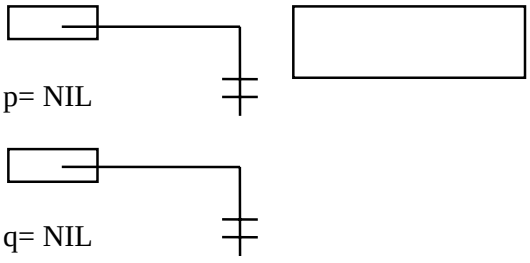
- al copiarse en p la dirección de q, p y q quedan apuntando ambas a la variable dinámica que estaba apuntada por q , siendo ahora p^ = q^

- la variable dinámica p^ que se tenía antes de la asignación p := q, queda inaccesible después de la asignación, pero sin liberarse la memoria que ocupaba.

b) Si se tenía en cambio la siguiente situación antes de la asignación en cuestión

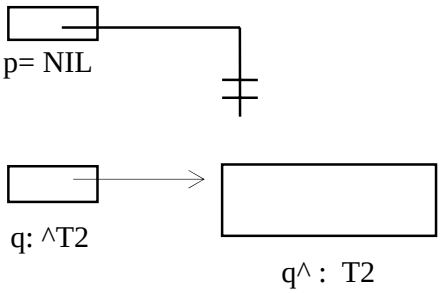


Luego de p := q quedará:

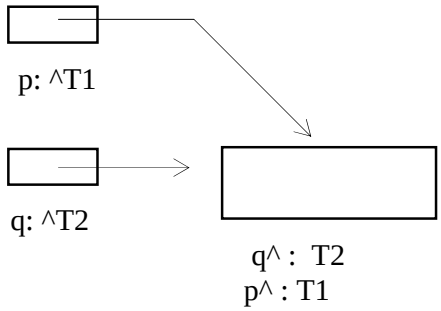


Observemos que al copiarse en p el valor NIL de q, p y q quedan ambas con valor nulo y la variable dinámica que estaba antes de la asignación apuntada por p queda inaccesible pero sin liberarse la memoria que ocupaba.

c) Por último, si la situación anterior a la asignación era:



Al ejecutarse p := q quedará:



Observación: Todas aquellas situaciones donde quedan las variables dinámicas inaccesibles después de la asignación p := q, deberían ser destruidas en un paso previo a la asignación usando el procedimiento DISPOSE con el propósito de hacer un buen manejo del heap de la memoria.

Asignaciones de variables dinámicas

Si una variable dinámica p^ es de tipo T, el manejo que se puede hacer de ella es exactamente igual al que se puede hacer con las variables no dinámicas del mismo tipo.

Por lo tanto, si se tienen las variables dinámicas p^ : T y q^ : T y si es válida la asignación entre variables de tipo T[4], son válidas también las asignaciones:

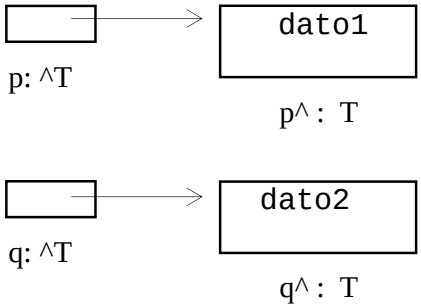
p^ := q^ y q^ := p^

Al ejecutarse p^ := q^ , tanto p^ como q^ quedarán con el mismo valor (el que previamente tenía q^)

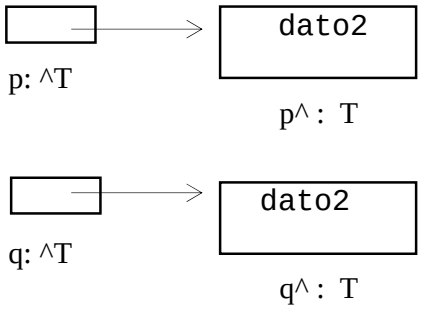
Gráficamente:

[4] Observar que si T es de tipo file, o tiene definidas componentes de tipo file, la asignación no es posible.

a) Si se tenía :

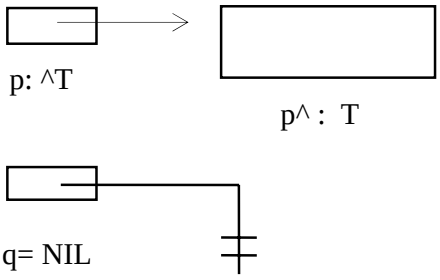


Luego de p^ := q^ quedará:



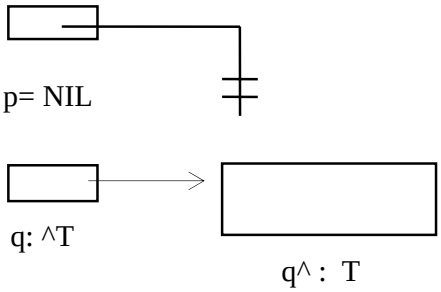
Observemos que al copiarse en p^ el contenido q^, quedan ambas variables vigentes y con el mismo valor (el que previamente a la asignación tenía q^).

b) Si se tenía en cambio la siguiente situación antes de la asignación en cuestión



Al ejecutarse p^ := q^ se produce un ERROR en tiempo de ejecución (es decir el compilador no detectará el error) pues se intenta acceder a la variable q^ para obtener su valor, pero dicha variable no existe pues el valor de q es nulo.

c) Por último, si la situación anterior a la asignación era:



Al ejecutarse p^ := q^ se produce ERROR en tiempo de ejecución por análoga razón que en el punto b)

Los punteros como herramientas para la construcción de estructuras dinámicas

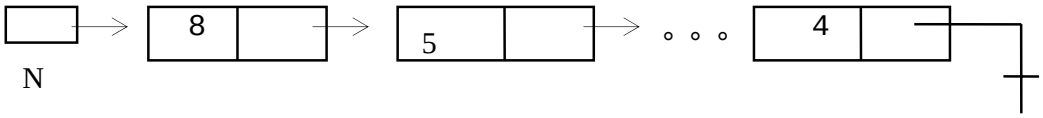
Los punteros constituyen una herramienta de gran valor para construir estructuras dinámicas. La posibilidad de crear estas estructuras usando punteros, se debe además a la siguiente situación: las variables dinámicas pueden ser conglomerados donde uno o más campos pueden ser de tipo puntero. Esto permite crear variables dinámicas que en algunos de sus campos guardarán direcciones de otras variables dinámicas, enlazando de esta manera variables dinámicas que conforman una estructura dinámica.

Observemos entonces que no todos los punteros deben ser declarados. Cuando un puntero es componente de una variable dinámica no se declara sino que se crea conjuntamente con todos los campos de la variable dinámica.

Ejemplos

I) Para representar **números enteros no negativos largos** (de gran cantidad de dígitos, donde ese número de dígitos no se puede acotar por alguna razón), puede pensarse en una estructura dinámica compuesta de celdas donde cada una de ellas aloja un dígito y un puntero a la próxima celda.

Gráficamente:



- La estructura puede definirse de la siguiente manera:

```
const   Fin = -1;

type    Digito =  -1..9   (* incluye el valor de la constante Fin *)
```

```
Celda = Record
            cifra : Digito;
            enlace : ^ Celda
        End;

var     p, Numero :  ^ Celda;
        Dto : Digito;
```

La constante Fin permite hacer la entrada de todos los dígitos, entrando el valor -1 (que no corresponde a ningún dígito) para terminar.

El número que se entra queda en una estructura enlazada a la cual se puede ingresar a través de la variable Numero.

- El algoritmo para la creación de una estructura dinámica de un número entero largo podría ser:

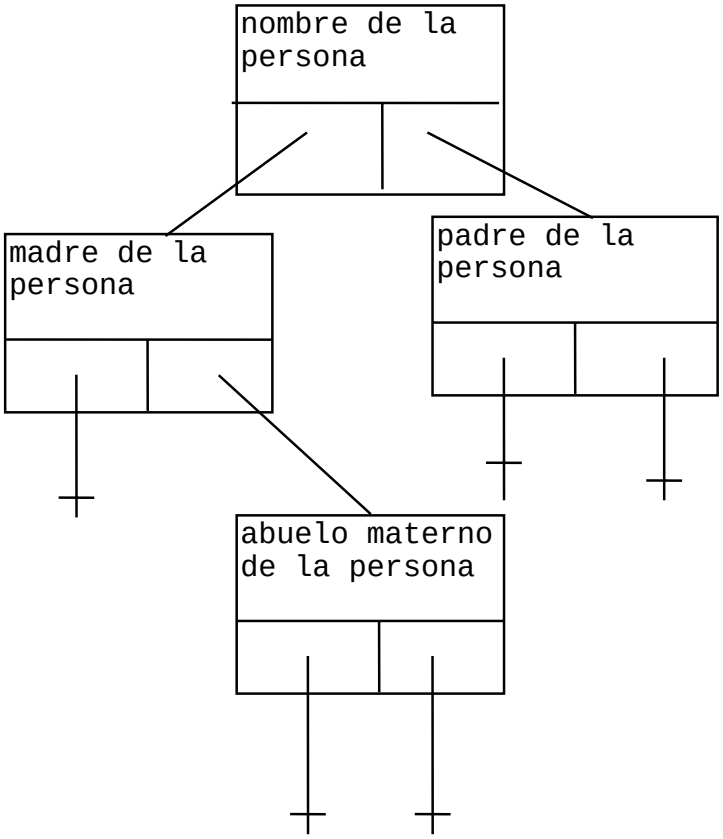
```
leer (Dto);
si  Dto <> Fin entonces
        NEW(p);
        Numero := p;

mientras  Dto <> Fin  hacer
    p^. cifra := Dto;
    leer (Dto);
    si Dto <> Fin  entonces
            NEW(p^. enlace);
            p := p^. enlace
    si no
        p^. enlace := NIL
```

II) La **genealogía de una persona** condensa la información sobre sus ancestros.

La genealogía de una persona, puede definirse mediante: el nombre de la persona, la genealogía de la madre y la genealogía del padre. Para establecer la genealogía de una persona debe conocerse su nombre, pero puede ocurrir que no se conozca la genealogía de su padre y/o la de su madre.

Una estructura adecuada para representar la genealogía de una persona podría ser la que se presenta en el siguiente esquema:



- La estructura propuesta puede definirse de la siguiente manera:

```
Type   Celda = record
                  nombre: Cadena;  (* Cadena es un tipo declarado previamente  *)
                  padre,           (* para representar palabras               *)
                  madre : ^ Celda
               End;
        Genealogia = ^ Celda
```

- Para crear la genealogía de una persona , se puede recurrir a un procedimiento como el siguiente:

```
procedimiento CrearGenealogia (NN : Cadena; Var p : Genealogia);

  NEW (p);
  p^. nombre := NN;
  si Conocido ( NombreMadre , NN ) entonces
                                    CrearGenealogia (NombreMadre , P^. madre )
```

```
sino
    P^. Madre := NIL

si Conocido ( NombrePadre , NN ) entonces
                                  CrearGenealogia (NombrePadre , P^. padre )
sino
    P^. Padre := NIL
```

La función Conocido que invoca CrearGenealogia, devuelve un valor booleano según se conozca o no la madre (el padre) de NN, y en caso de que sea conocido, NombreMadre (NombrePadre) trae la cadena correspondiente.