Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 2: MEMORIA DINÁMICA y PUNTEROS

MEMORIA DINÁMICA y PUNTEROS

- Introducción
- Conceptos básicos
- Definición y declaración de punteros
- Creación y destrucción de variables dinámicas.
- Operaciones básicas con datos referenciados
- Operaciones básicas con punteros
- El valor nil
- Aplicaciones no recursivas de los punteros

INTRODUCCIÓN

- Las estructuras de datos hasta ahora vistas se almacenan estáticamente en la memoria física del ordenador.
  - El espacio de memoria se reserva con anticipación y no cambia durante la ejecución del programa*.
  - Esto permite una comprobación de tipos en tiempo de compilación.
- Inconvenientes de la configuración estática:
  - Su rigidez, ya que estas estructuras no pueden crecer o menguar durante la ejecución del programa.

* Esto no implica que la cantidad de memoria de ejecución de un programa sea constante, ya que dependerá del numero de subprogramas recursivos invocados por el programa.

INTRODUCCIÓN

- La definición y manipulación de estos objetos se realiza en Pascal mediante los **punteros** (variables cuyo contenido son posiciones de memoria).

- Ventaja frente a las estructuras estática:
  - La **flexibilidad** que poseen las estructuras dinámicas en cuanto a las formas que pueden adoptar: árboles, listas, redes, etc...

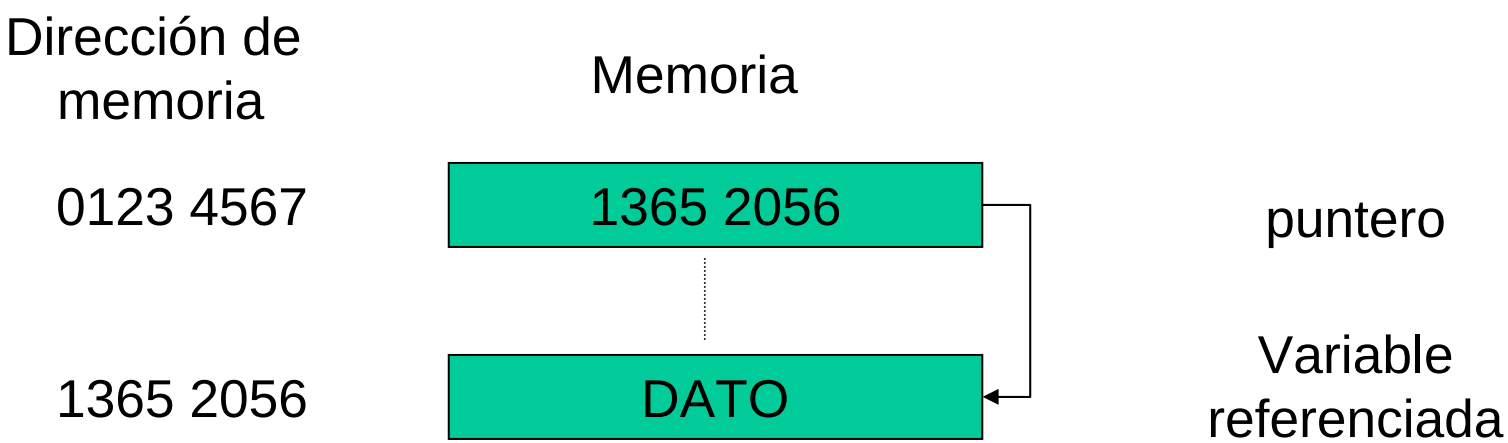
- Inconvenientes:
  - **Alliasing**: Doble direccionamiento sobre una misma variable lo que implica efectos laterales.

  - **Gestión de la memoria**: Su uso requiere una especial atención de la memoria disponible así como de la que ya no queramos utilizar.

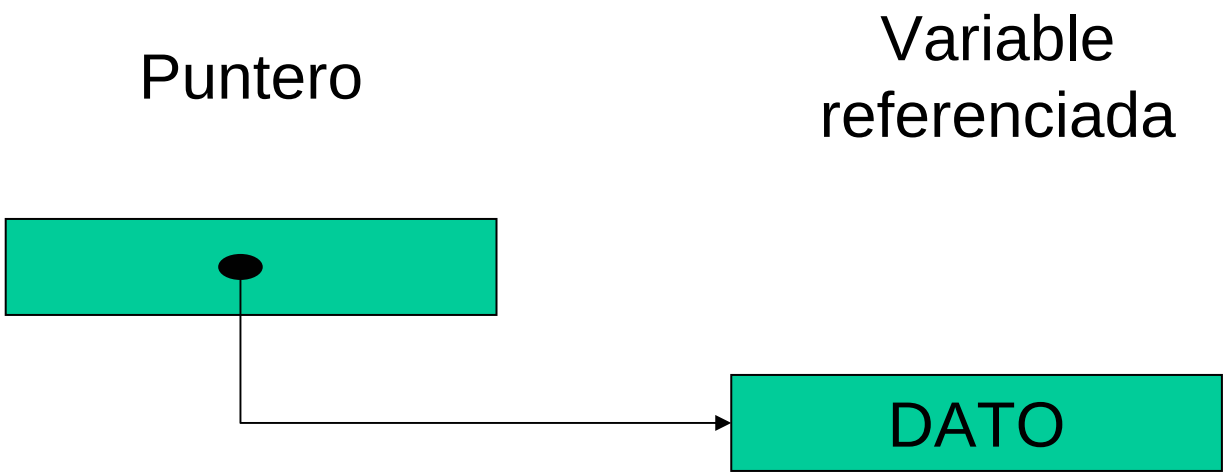
CONCEPTOS BÁSICOS

- **Un puntero** es una variable que contiene la dirección de memoria donde se encuentra almacenado un dato.

- **Una variable referenciada o dato apuntado** es el dato cuya posición en memoria está contenida en un determinado puntero (variable dinámica).



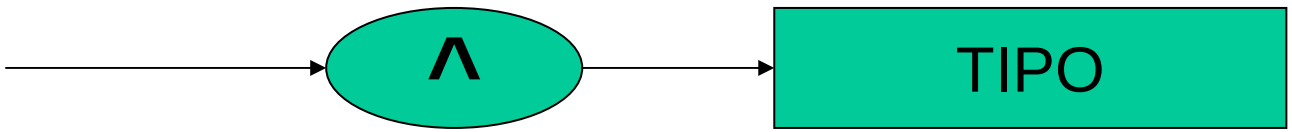
REPRESENTACIÓN GRÁFICA



DEFINICIÓN Y DECLARACIÓN DE PUNTEROS

- Para poder usar una variable puntero es necesario:
  - Definir el tipo de dato (o estructura) al que se apunta. (esta declaración se realiza dentro de la sección TYPE).
  - Declarar las variables punteros que sean necesarias (esta declaración se realiza dentro de la sección VAR).
- En Pascal un puntero sólo puede señalar a objetos de un mismo tipo, el establecido en la declaración.

DIAGRAMA SINTÁCTICO



Ejemplo I:

```
TYPE
        tapunchar=^char;
VAR
        apcar:=tapunchar;
```

Ejemplo II:

```
TYPE
        tApNodo=^tNodo;
        tNodo=record
                info:.....
                Sig:tApNodo
        end;
VAR
        ApNodo:=tApNodo;
```

ALGUNAS OBSERVACIONES AL RESPECTO

- Una variable de tipo puntero ocupa una cantidad de memoria fija, independiente del tipo de dato al que apunta.

- Un dato referenciado, como el del ejemplo, no posee existencia inicial, o lo que es lo mismo no existe inicialmente espacio reservado en memoria para el.

LA NECESIDAD DE UTILIZAR PUNTEROS

- Para poder emplear variables dinámicas es necesario emplear un tipo de dato que permita referenciar nuevas posiciones de memoria que no han sido declaradas a priori y que se van a crear y destruir en tiempo de ejecución.

- Estas variables son los punteros que en Pascal es un tipo de dato simple.

CREACIÓN Y DESTRUCCIÓN DE VARIABLES DINÁMICAS

- Las variables dinámicas son por definición aquellas que se crean cuando se necesitan y se destruyen cuando ya han cumplido con su cometido.

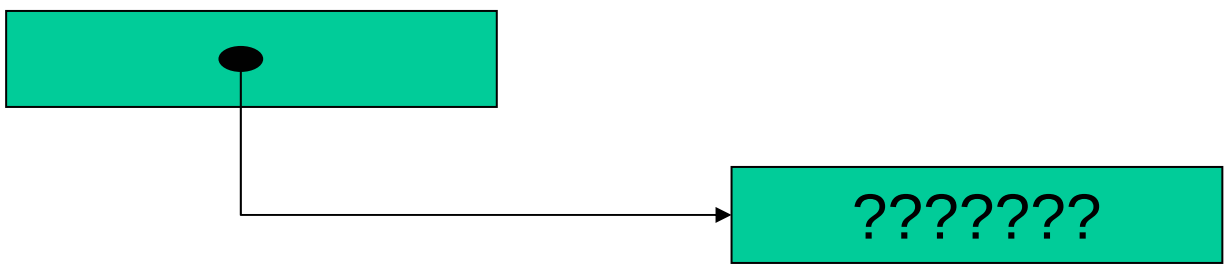
- En pascal la creación y destrucción de variables dinámicas se realiza mediante los siguientes procedimientos:

  - **New(puntero)**

  - **Dispose(puntero)**

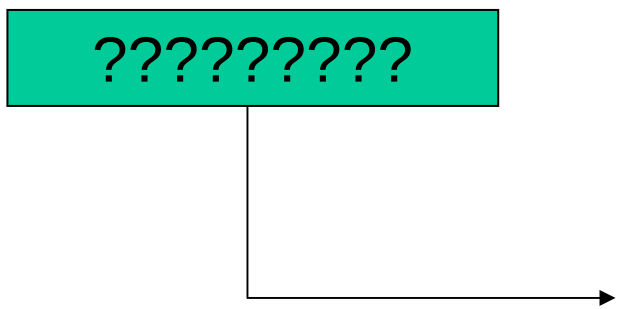
CREACIÓN DE UNA VARIABLE DINÁMICA

- New(puntero)
  - Reserva la memoria necesaria para un dato del tipo apropiado.
  - Coloca la dirección de memoria de esta nueva variable en el puntero.
- Gráficamente esto se representa:



DESTRUCCIÓN DE UNA VARIABLE DINÁMICA

- Dispose(puntero)
  - Libera la memoria asociada a la variable referida (dejándola libre para otros fines).
  - Deja indefinido el valor del puntero.
- Gráficamente esto se representa:



OPERACIONES BÁSICAS CON VARIABLES REFERENCIADAS

- El contenido de la variable referenciada por el puntero se denota:

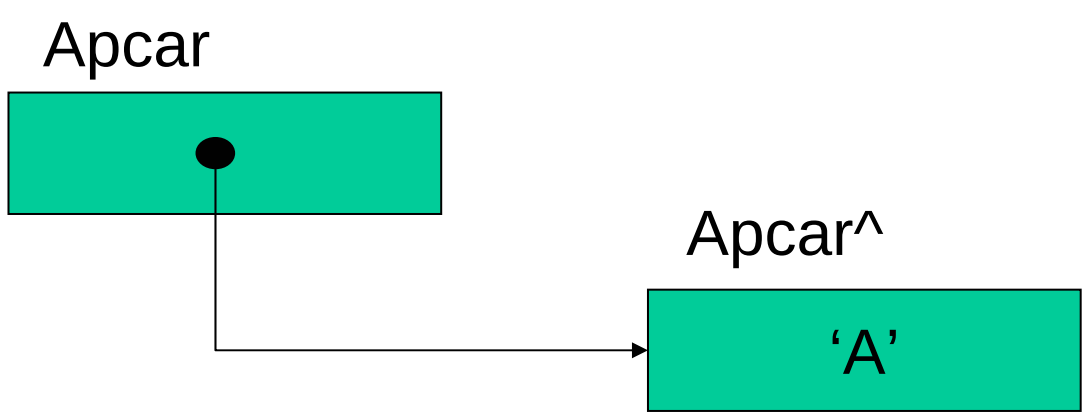
  puntero^

- Las operaciones permitidas para esta nueva variables son:

  - Asignación
  - Lectura
  - Escritura
  - Todas las operaciones legales que se puedan realizar con dicho tipo.

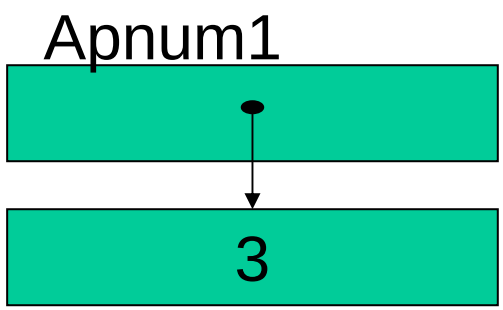
EJEMPLO I

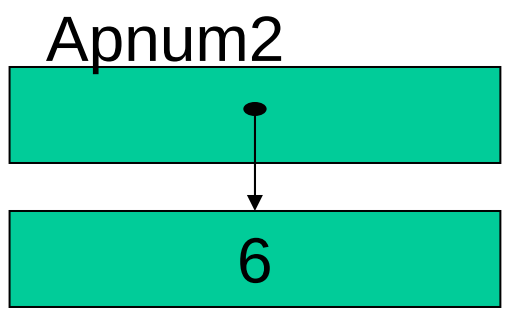
```
.....
TYPE
    tApcar=^char;
VAR
    Apcar:tApcar;

BEGIN
.....
New(Apcar);
Readln(Apcar^); {Por ejemplo ‘B’}
Apcar^:=Pred(Apcar^);
Writeln(Apcar^);
.....
END.
```



EJEMPLO II

```
.....
TYPE
    tApnum=^integer;
VAR
    Apnum1, Apnum2:tApnum;
BEGIN
.....
New(Apnum1); New(Apnum2);
Apnum1^:=2; Apnum2^:=4;
Apnum2^:=Apnum1^+Apnum2^;
Apnum1^:=Apnum2^ DIV 2;
.....
END.
```



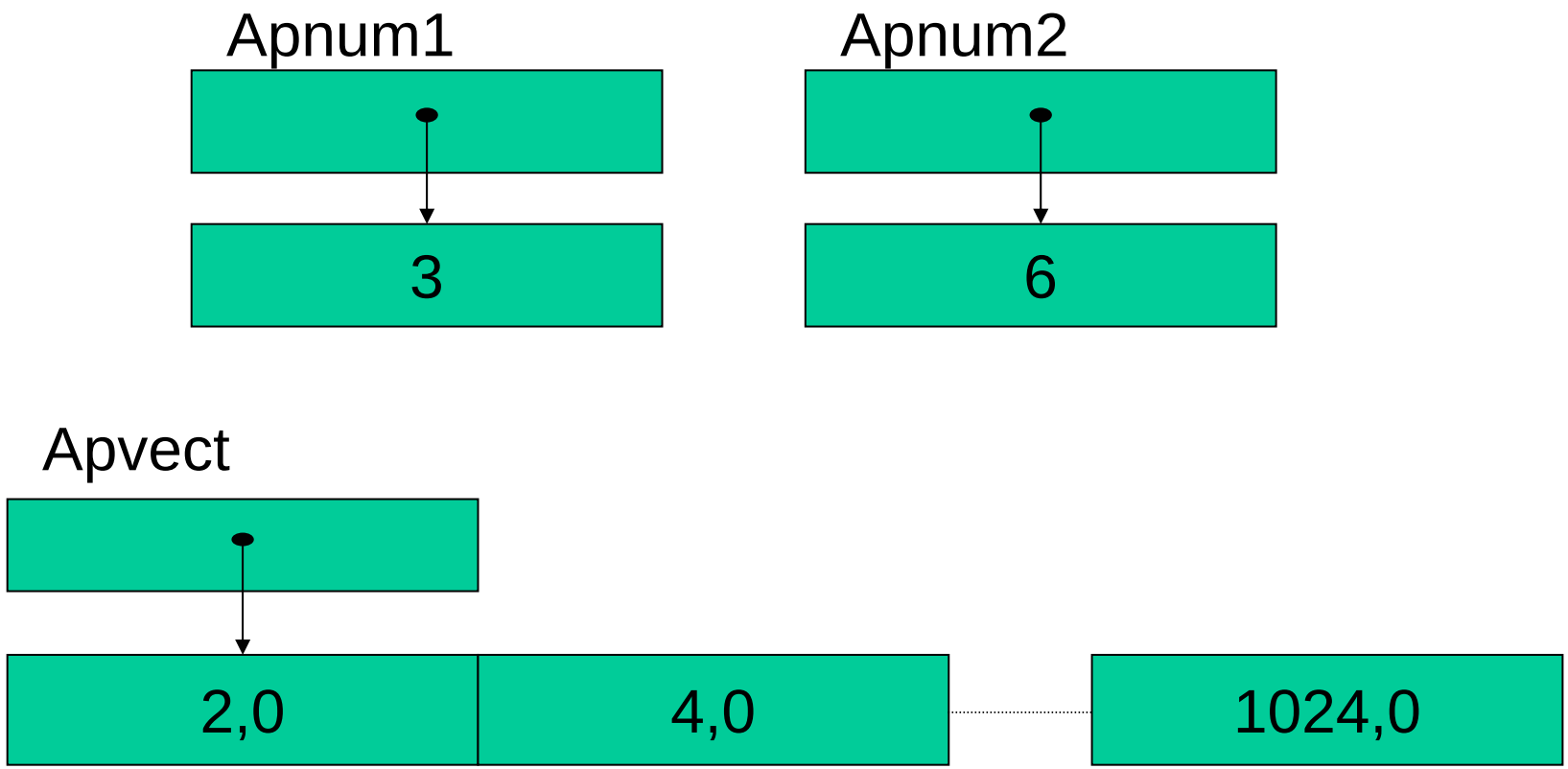


EJEMPLO III

```
.....
TYPE
    tVector10=array[1..10] of real;
    tApnumero=^integer;
    tApvector=^tvector10;
VAR
    Apnum1, Apnum2: tApnum;
    Apvect: tApvector10;
    i: integer;
BEGIN
.....
    New(Apnum1); New(Apnum2); New(apvect);
    Apnum1^:=45; Apnum2^:=30;
    Apvect^[1]=2;
    for i:=2 to 10 do
            Apvect^[i]:=Apvect^[i-1] * 2;
.....
END.
```

EJEMPLO III

- Dejando el estado de la memoria de la siguiente forma:



OPERACIONES BÁSICAS CON PUNTEROS

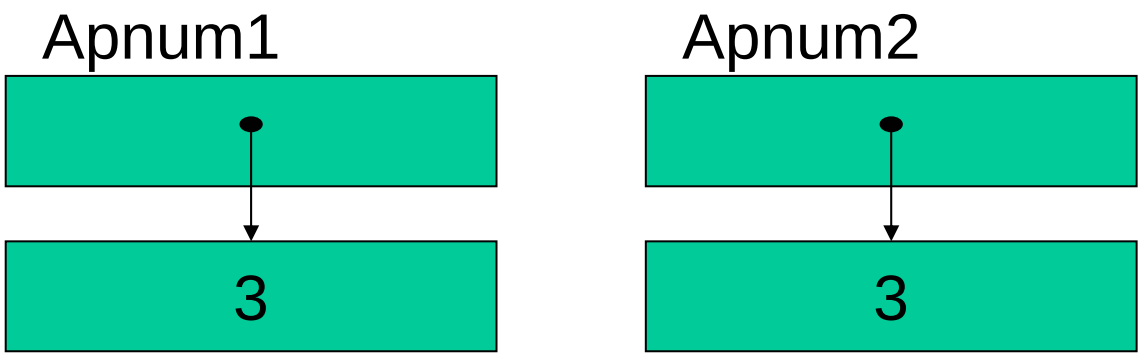
- Las únicas operaciones válidas son:
  - **La comparación** (se comparan las direcciones, no los contenidos de los datos apuntados).

    Apnum1=Apnum2

  - **La asignación** (se asignan las direcciones entre sí, no los contenidos de los datos apuntados).

    Apnum1:=Apnum2

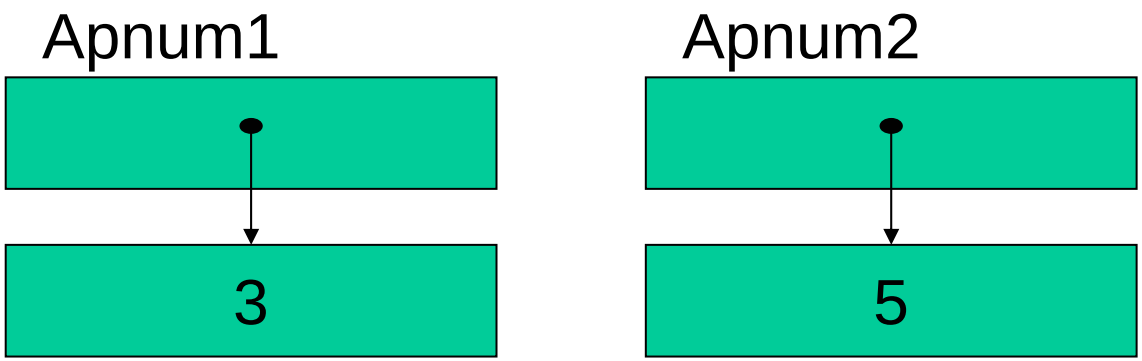
LA COMPARACIÓN



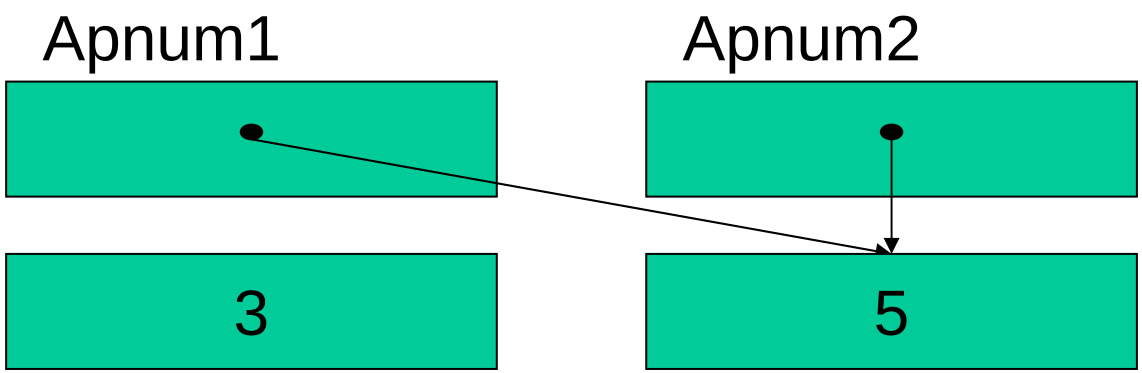
- Apnum1=Apnum2

- La comparación anterior daría como resultado el valor ‘false’ ya que cada uno apunta a una dirección de memoria diferente.

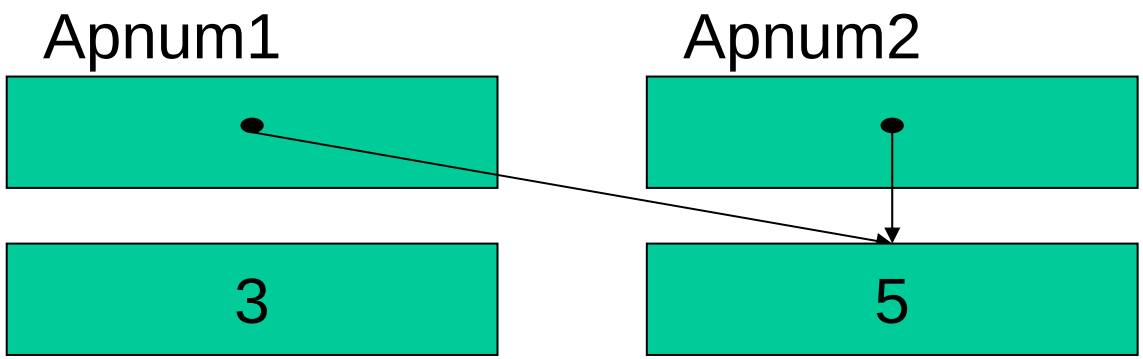
LA ASIGNACIÓN



- Apnum1:=Apnum2



LA ASIGNACIÓN



- Los cambios efectuados sobre Apnum1 afectan a la variable Apnum2 (son indistintas) (Alliasing).

- El espacio de memoria reservado inicialmente por el puntero Apnum1 sigue situado en memoria. Una adecuada **gestión de la memoria** hubiera exigido la liberación de ese espacio antes de efectuar la asignación.

CONSISTENCIA ENTRE TIPOS

- Operaciones válidas
  - Apnum1:=Apnum1
  - Apnum1=Apnum2
  - Apvector1:=Apvector2
- Operaciones no válidas:
  - Apnum1:=Apchar;
  - Apnum1=Apvector;

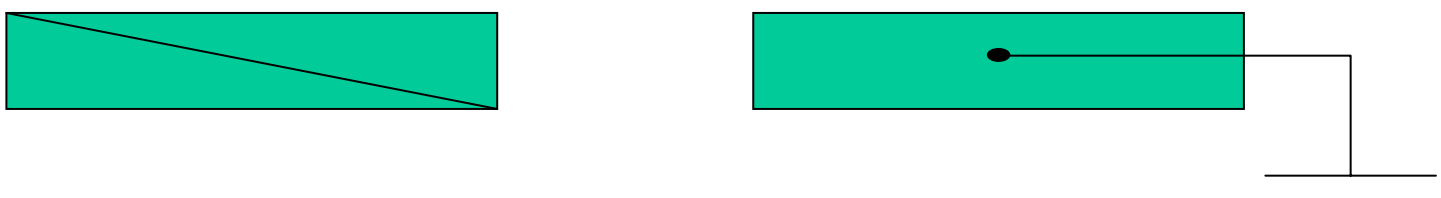
EL VALOR NIL

- Un modo alternativo de asignar un valor a un puntero es indicar que no apunta a ningún dato. Esto se lleva a cabo mediante la constante predefinida "**nil**"

- "**nil**" es independiente del tipo del dato apuntado por lo que puede ser utilizado por cualquier puntero.

  - Apcar:=nil;
  - Apcar=nil;

- Representación gráfica:



APLICACIONES NO RECURSIVAS CON PUNTEROS

- Asignación de datos compuestos en un solo paso

- Definición de funciones que devuelven datos compuestos.

ASIGNACIÓN DE DATOS COMPUESTOS

- Esta aplicación es de utilidad cuando se manejan variables o estructuras de datos de gran tamaño (por el elevado coste que supone el realizar la copia de todas sus componentes).
  - Asignación de variables de gran tamaño
  - Ordenación de vectores con elementos de gran tamaño.

ORDENACIÓN DE VECTORES CON ELEMENTOS DE GRAN TAMAÑO

```
TYPE
tApFicha=^tFicha;
tFicha=record
    nombre:string;
    direccion:string;
    ..............
End; {tFicha}
{tListaAlumnos=Array[1..100] of tFicha;}
tlistaApAlum=array[1..100] of tApFicha;
.............
```

- Las operaciones de ordenación y búsqueda se realizan sobre el array de punteros.

FUNCIONES QUE DEVUELVEN DATOS COMPUESTOS

- Como en el caso anterior se cambia el objeto por el puntero que lo apunta.

- Ejemplo: Definir un subprograma que a partir de un punto del plano, un ángulo y una distancia calcule la posición de un nuevo punto.

FUNCIONES DE RESULTADO NO SIMPLE, EJEMPLO

```
.....
TYPE
    tPunto=record
            x,y:real;
    end; {tPunto}
    tApPunto=^tPunto;

VAR
    angulo,distancia:real;
    origen:tPunto;
    pDestino:tApPunto;
.....
FUNCTION Destino(orig:tPunto;ang,dist:real):tApPunto
```

FUNCIONES DE RESULTADO NO SIMPLE, EJEMPLO

```
FUNCTION Destino(orig:tPunto;ang,dist:real):tApPunto
VAR
    pPun:tApPunto;
Begin
    New(pPun);
    pPun^.x:=orig.x+dist*cos(ang);
    pPun^.y:=orig.y+dist*sen(ang):
    destino:=pPun;
End; {Destino}
```