TEMA 0
Gestión de Memoria Dinámica

ESTRUCTURAS DE DATOS

Objetivos

- Tema preliminar para entender el uso de la herramienta básica en la gestión de memoria dinámica: punteros

- **Objetivos:**
  - Conocer el concepto de “puntero”
  - Entender la gestión dinámica de memoria
  - Manejar estructuras estáticas y dinámicas en memoria a través de punteros
  - Crear y destruir estructuras dinámicas en memoria

Definición del problema

- Las estructuras estáticas (por ejemplo, array) no pueden cambiar su tamaño durante la ejecución del programa
- Cambiar la disposición de los elementos dentro de la estructura estática es, a veces, costoso.
- Ejemplos:
  - No se puede redimensionar un array.
  - Colocar el último elemento al comienzo del array.
- Además, hay otros factores importantes a tener en cuenta sobre el uso de la memoria en los procesos.

Definición del problema

- El espacio de memoria en un sistema está descompuesto de forma general en 4 bloques con tamaños diversos
  - Segmento de código: asignación **automática**
  - Variables globales: asignación **automática**
  - *Stack* o pila de memoria: asignación **automática**
  - *Heap* o montículo de memoria: asignación **manual**

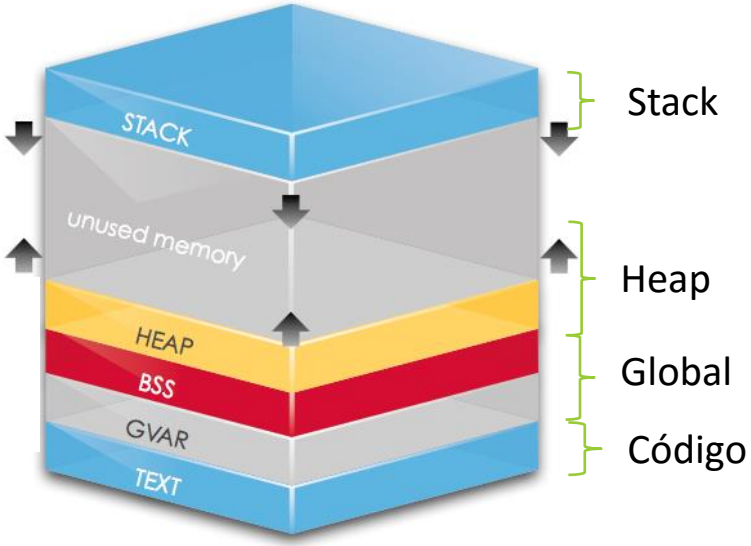


Imagen extraída de: www.sw-at.com

Definición del problema

- La memoria local a los subprogramas se gestiona en la pila de memoria
  - Cada proceso de un programa tiene su propia pila de memoria, por lo que en general la pila tiene un tamaño muy limitado
- La memoria dinámica se gestiona en un bloque muy grande de memoria (*heap*/montículo)

```
PROGRAM Memoria;
VAR
  precio: real;

FUNCTION CalculoIVA(p: real): real;
BEGIN
  CalculoIVA := p*0.21;
END;

BEGIN
  precio := 200.25;
  writeln('El IVA es: ', CalculoIVA(precio));
END;
```

global

local

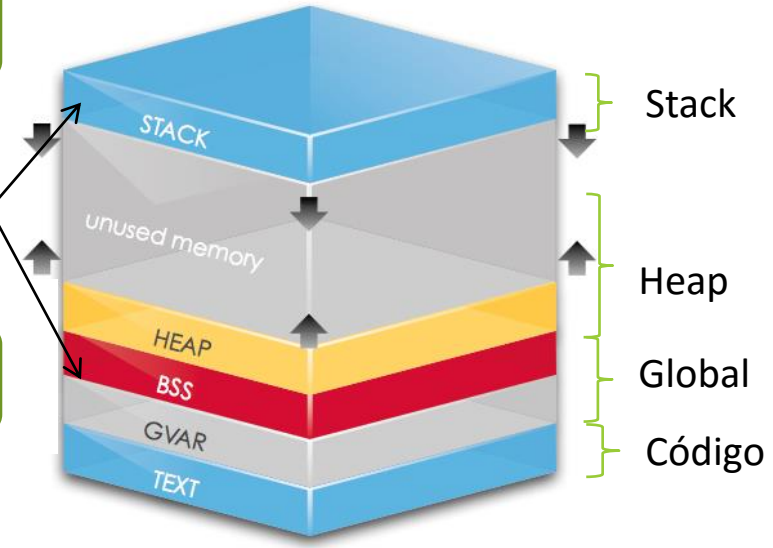


Imagen extraída de: www.sw-at.com

Definición del problema

- Para algunos problemas de programación no se conoce en tiempo de diseño cuánta memoria necesitaremos ni cómo se va a organizar

- **Solución**: definir y organizar esa memoria en tiempo de ejecución
  - Para ello, se utilizan estructuras de memoria dinámica

- La gestión de memoria dinámica se realiza a través de variables capaces de guardar direcciones de memoria: **punteros**

¿Qué es eso de…?

- **Memoria dinámica**: memoria en la que se puede reservar espacio en tiempo de ejecución
  - El *heap* es el bloque del espacio direccionable de memoria dedicado para la memoria dinámica

- **Estructuras de datos dinámicas**: colección de elementos (denominados nodos) que se crean o destruyen en tiempo de ejecución.

- **Variables Dinámicas**: Posiciones de memoria reservadas en tiempo de ejecución

Punteros

- Una variable puntero se puede declarar como tipo anónimo, o como tipo definido por el usuario

```
VAR
 pointer: ^integer;
```

Variable puntero de tipo anónimo identificada por `pointer`

```
TYPE
  TPointer = ^integer;
VAR
  pointer: TPointer;
```

Tipo `TPointer` **definido por el usuario** para crear variables puntero a entero

Variable declarada del tipo anterior

Punteros

- Un tipo puntero se puede usar para declarar variables de ese tipo
  - Igual que un tipo Entero se usa para declarar variables de tipo entero (que guarda valores de ese tipo)

```
TYPE
  TPrecio = integer;
VAR
  precioPan: TPrecio;
BEGIN
  precioPan := 85;
  writeln(precioPan);
END.
```

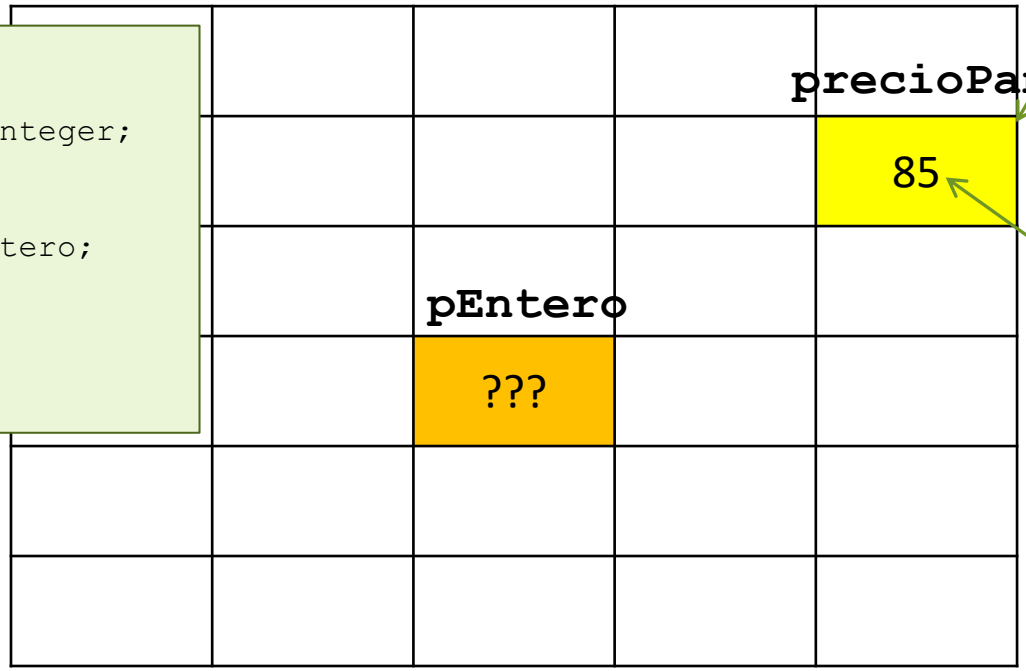
```
TYPE
  TPunteroEntero = ^integer;
VAR
  pEntero: TPunteroEntero;
```

  - Una variable puntero almacena una dirección de memoria donde guardar un valor del "tipo base" del puntero (**releer hasta estar bien seguro de entenderlo**)

Punteros

- Simulación en memoria

```
TYPE
  TPrecio = integer;
  TPunteroEntero = ^integer;
VAR
  precioPan: TPrecio;
  pEntero: TPunteroEntero;
BEGIN
  precioPan := 85;
  writeln(precioPan);
END.
```



Dirección $3$12 identificada como `precioPan`

Valor (85) que guarda `precioPan`

Operaciones con punteros

- Operador @ (Referencia)
  - Obtención de la dirección de memoria de una variable

```
VAR
  pEntero: ^integer;
  precioPan: integer;
BEGIN
  precioPan := 85;
  pEntero   := @precioPan;
  ...
```

- Operador ^ (Desreferencia)
  - Acceso al valor de la variable apuntada desde el puntero

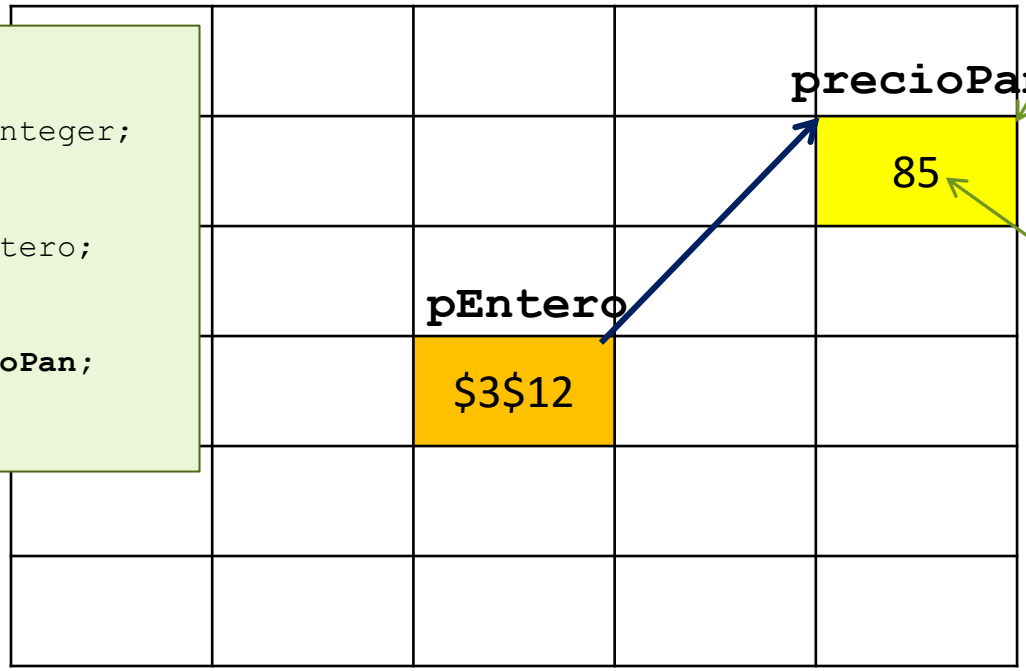
```
  ...
  pEntero^ := 100;
  writeln(precioPan); {imprime 100}
```

`pEntero^` y `precioPan` son **sinónimos**

Punteros

- Simulación en memoria

```
TYPE
  TPrecio = integer;
  TPunteroEntero = ^integer;
VAR
  precioPan: TPrecio;
  pEntero: TPunteroEntero;
BEGIN
  precioPan := 85;
  pEntero   := @precioPan;
  writeln(precioPan);
END.
```



Dirección $3$12 identificada como `precioPan`

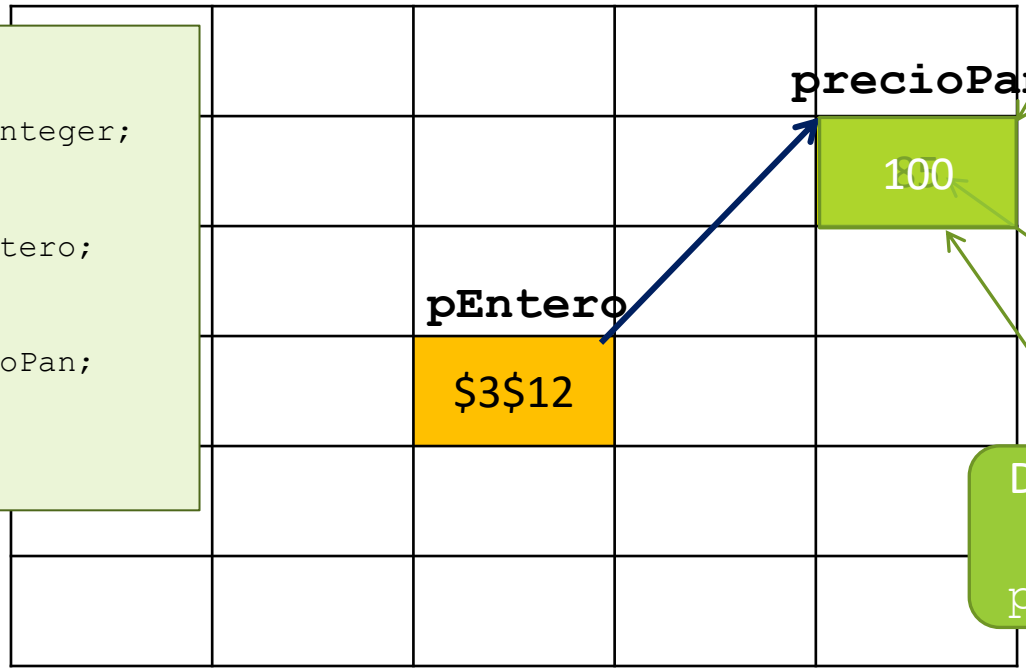
Valor (85) que guarda `precioPan`

Si `pEntero` contiene el valor $3$12, y esa es la dirección de memoria de la variable `precioPan`, se dice que `pEntero` apunta a `precioPan`

Punteros

- Simulación en memoria

```
TYPE
  TPrecio = integer;
  TPunteroEntero = ^integer;
VAR
  precioPan: TPrecio;
  pEntero: TPunteroEntero;
BEGIN
  precioPan := 85;
  pEntero   := @precioPan;
  pEntero^  := 100;
  writeln(precioPan);
END.
```



Dirección $3$12 identificada como `precioPan`

Valor (85) que guarda `precioPan`

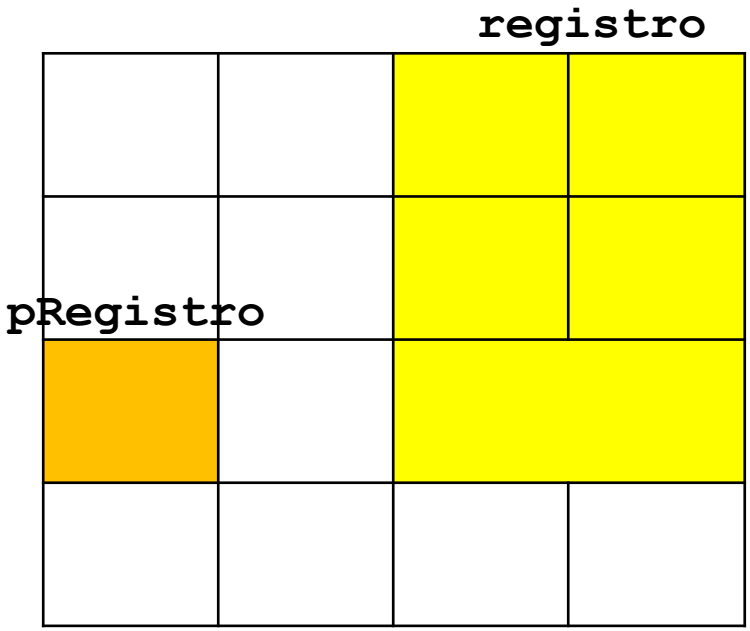
Desde `pEntero` se altera el valor de `precioPan` (100)

Si `pEntero` contiene el valor $3$12, y esa es la dirección de memoria de la variable `precioPan`, se dice que `pEntero` apunta a `precioPan`

Operador @: Ejemplo (1/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  registro: tRegistro;
BEGIN
  ...
  registro.iniciales:='ASM';
  registro.identificacion:=23455;
  ...
  pRegistro := @ registro;
  ...
END.
```

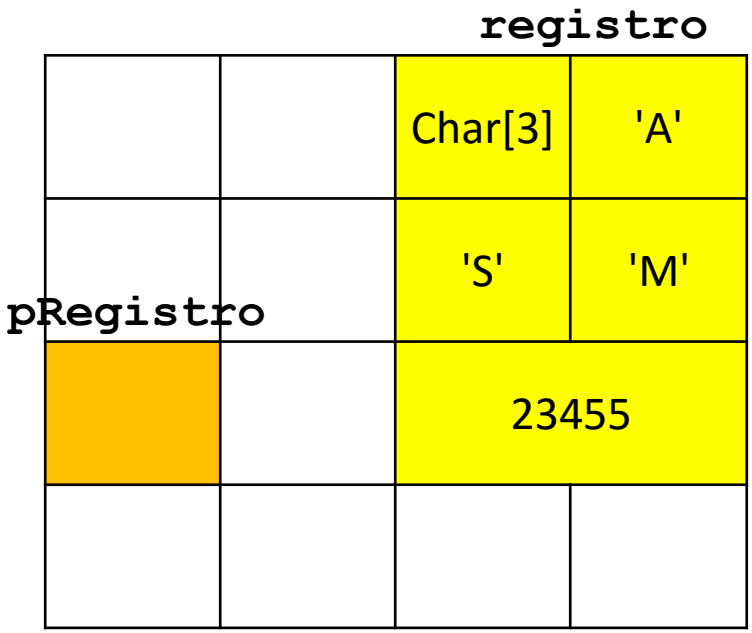
Reservamos en tiempo de compilación un bloque de memoria para un registro y otro para un puntero (ESTÁTICOS!!)



Operador @: Ejemplo (2/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  registro: tRegistro;
BEGIN
  ...
  registro.iniciales:='ASM';
  registro.identificacion:=23455;
  ...
  pRegistro := @ registro;
  ...
END.
```

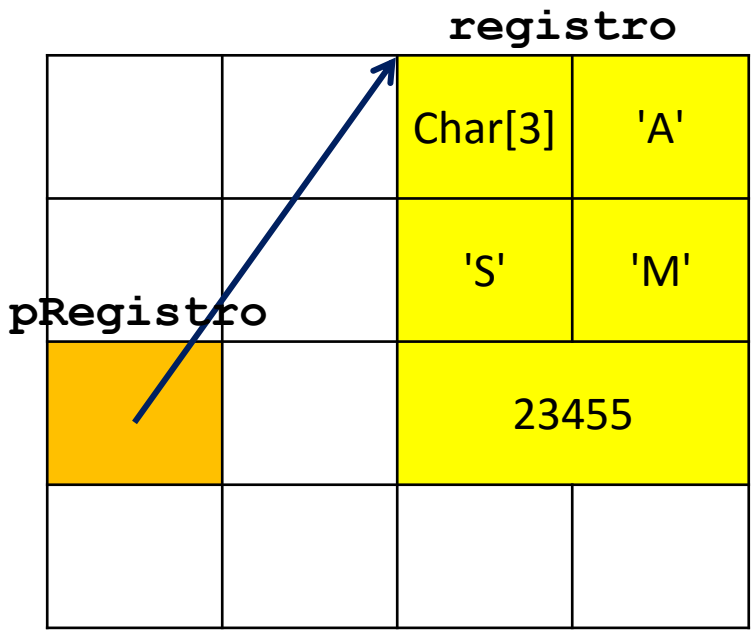
Inicializamos el registro de la manera habitual



Operador @: Ejemplo (y 3/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  registro: tRegistro;
BEGIN
  ...
  registro.iniciales:='ASM';
  registro.identificacion:=23455;
  ...
  pRegistro := @registro;
  ...
END.
```

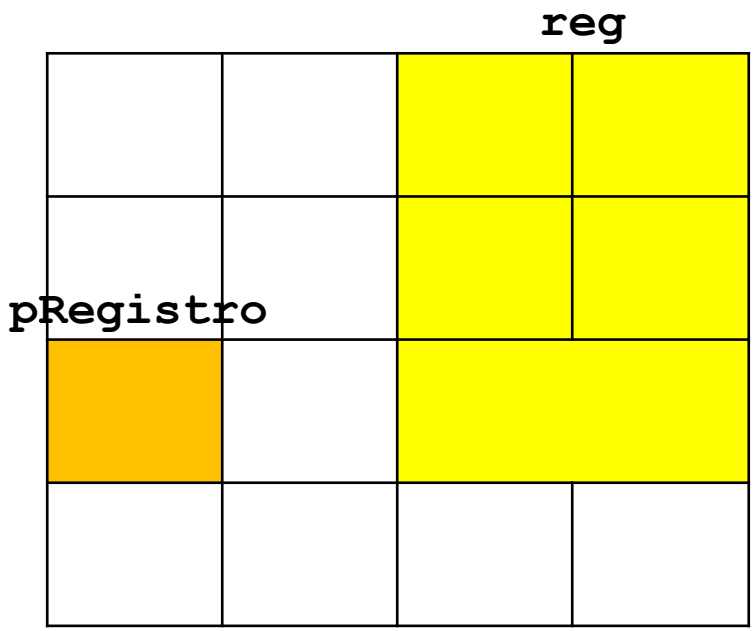
Apuntamos con el puntero el bloque de memoria del registro. Tenemos accesible la información del registro a través del puntero.



Operador ^: Ejemplo (1/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  reg: tRegistro;
BEGIN
  ...
  pRegistro:=@reg;
  …
  pRegistro^.iniciales:='JJP';
  pRegistro^.identificacion:=23456;
  ...
END.
```

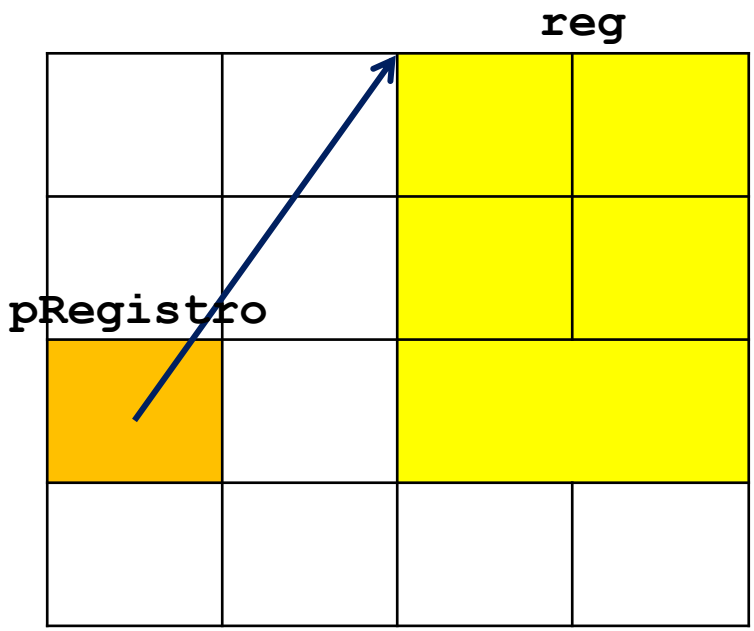
Reservamos en tiempo de compilación un bloque de memoria para un registro y otro para un puntero (ESTÁTICOS!!)



Operador ^: Ejemplo (2/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  reg: tRegistro;
BEGIN
  ...
  pRegistro:=@reg;
  …
  pRegistro^.iniciales:='JJP';
  pRegistro^.identificacion:=23456;
  ...
END.
```

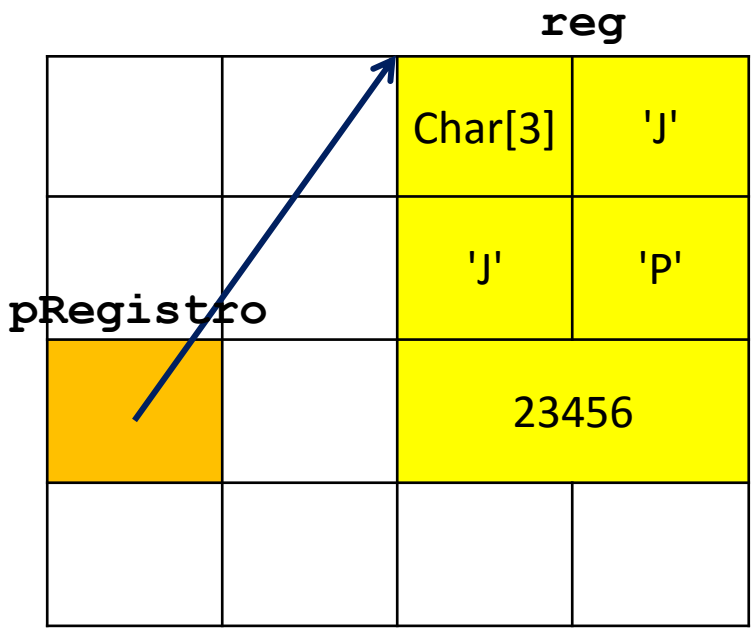
Apuntamos con el puntero el bloque de memoria del registro.



Operador ^: Ejemplo (y 3/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro= ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  reg: tRegistro;
BEGIN
  ...
  pRegistro:=@reg;
  …
  pRegistro^.iniciales:='JJP';
  pRegistro^.identificacion:=23456;
  ...
END.
```

Accedemos al campo "iniciales" e "identificacion"del dato de tipo `tRegistro` y asignamos valores (NO asignamos valores al puntero, sino al dato al que apunta)



Operaciones con Punteros

- Operaciones permitidas:
  - Asignación (:=)
  - Comparación (= y <>)

- Para realizar estas operaciones los operandos han de ser **punteros a variables del mismo tipo** o **NIL**.

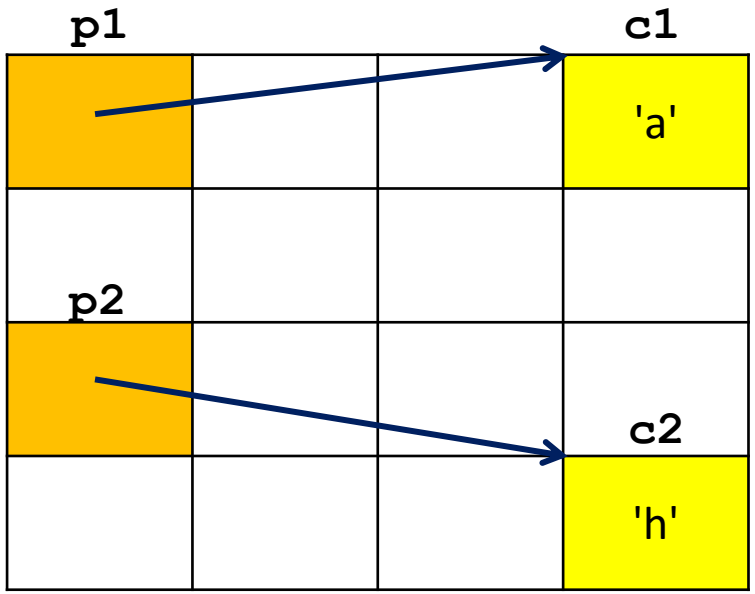
- Los punteros no se pueden leer ni escribir directamente.

Operaciones: ejemplos (1)

```
TYPE
  ptrACaracter = ^char;
VAR
  p1,p2: ptrACaracter;
  c1,c2: char;
BEGIN
  c1:='a';
  c2:='h';
  p1:=@c1;
  p2:=@c2;
  ...
END.
```

En esta situación:

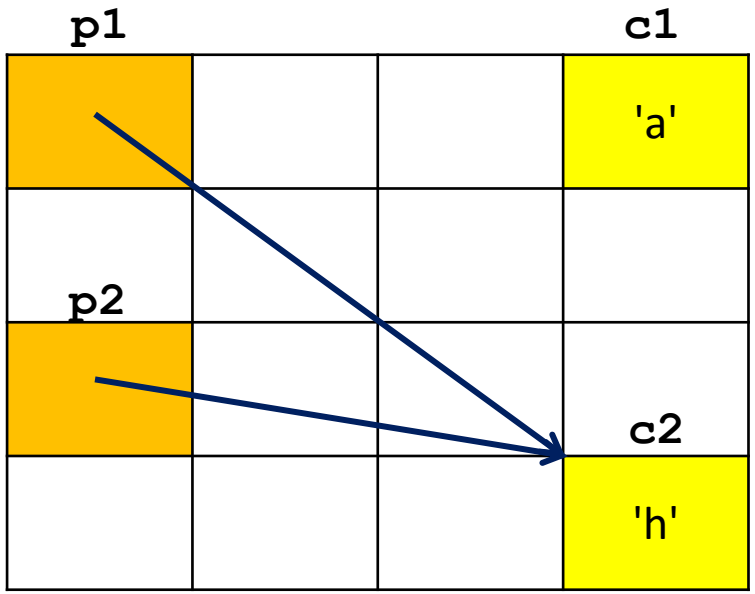
```
p1 = p2 --> FALSE
p1 <> p2 --> TRUE
p1^ = p2^ --> FALSE
```



Operaciones: ejemplos (2)

```
TYPE
  ptrACaracter = ^char;
VAR
  p1,p2: ptrACaracter;
  c1,c2: char;
BEGIN
  c1:='a';
  c2:='h';
  p1:=@c1;
  p2:=@c2;
  p1:=p2;
  ...
END.
```

Asignación de punteros.
En esta situación:
`p1 = p2 --> TRUE`
`p1 <> p2 --> FALSE`
`p1^ = p2^ --> TRUE`



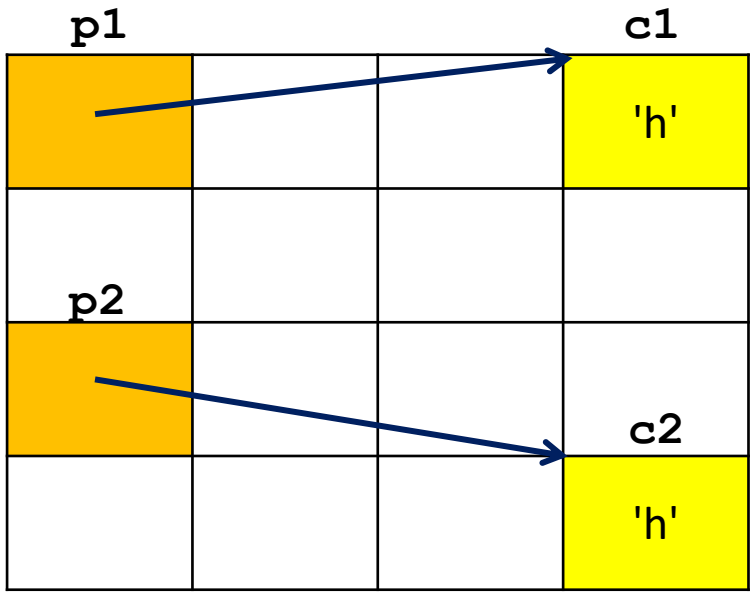
Operaciones: ejemplos (3)

```
TYPE
  ptrACaracter = ^char;
VAR
  p1,p2: ptrACaracter;
  c1,c2: char;
BEGIN
  c1:='a';
  c2:='h';
  p1:=@c1;
  p2:=@c2;
  p1^:=p2^;
  ...
END.
```

Asignación de valores

En esta situación:

```
p1 = p2 --> FALSE
p1 <> p2 --> TRUE
p1^ = p2^ --> TRUE
```



Procedimientos y funciones

- Pueden ser parámetros, por valor y por referencia, de los subprogramas.
- Se pueden devolver como resultado de una función.
- Aunque `f` sea una función que devuelva un puntero a un registro, no se permite:

```
f(parámetrosReales)^.campoDelReg;
```

- La sintaxis correcta es:

```
varPuntero:=f(parámetrosReales);
```

- y posteriormente sí se puede acceder al campo:

```
varPuntero^.campoDelReg;
```

Paso por valor de punteros

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  registro: tRegistro;

PROCEDURE miProc(dato: tPtrRegistro);
BEGIN
   ...
END;

BEGIN
  registro.iniciales:='ASM';
  registro.identificacion:=23455;
  pRegistro := @ registro;
  miProc(pRegistro);
  ...
END.
```

El procedimiento recoge una copia del puntero, que apunta al mismo lugar que el original.

Cualquier cambio en esa información se verá reflejada en la información apuntada por el puntero original.

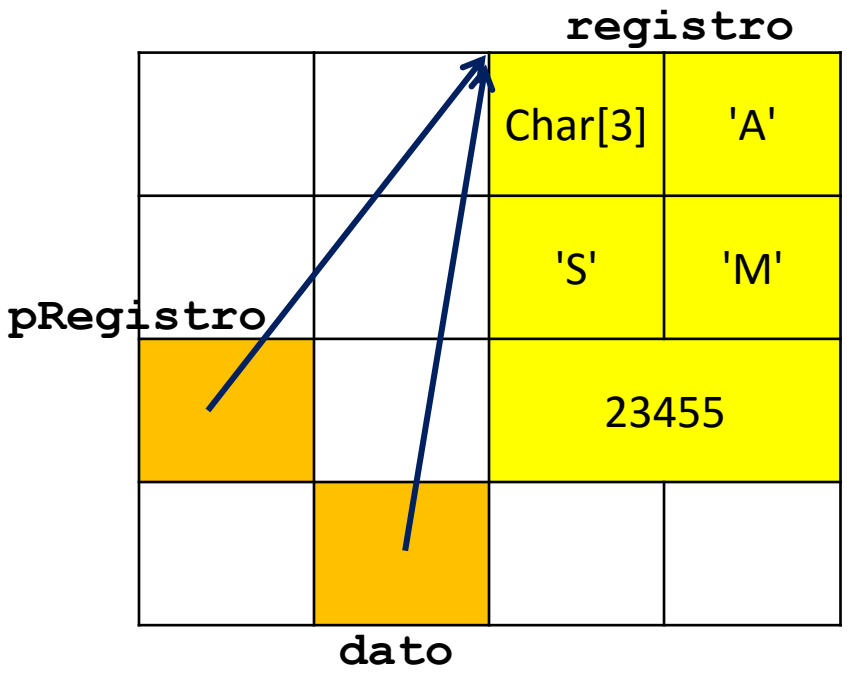
Paso por valor de punteros

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;
VAR
  pRegistro: tPtrRegistro;
  registro: tRegistro;

PROCEDURE miProc(dato: tPtrRegistro);
BEGIN
   ...
END;

BEGIN
  registro.iniciales:='ASM';
  registro.identificacion:=23455;
  pRegistro := @ registro;
  miProc(pRegistro);
  ...
END.
```

Cualquier cambio en esa información se verá reflejada en la información apuntada por el puntero original.



Operación **`new`**

- **`new`**: procedimiento por el que se **reserva un espacio de memoria dinámica**
- No siempre es obligatorio reservar memoria para utilizar un puntero (ver ejemplos anteriores)
- Sintaxis:

```
new(variablePuntero);
```

- Semántica:
  - reserva espacio de memoria para almacenar un dato del tipo base
  - y la variable puntero que se pasa como parámetro se deja apuntando a dicho espacio

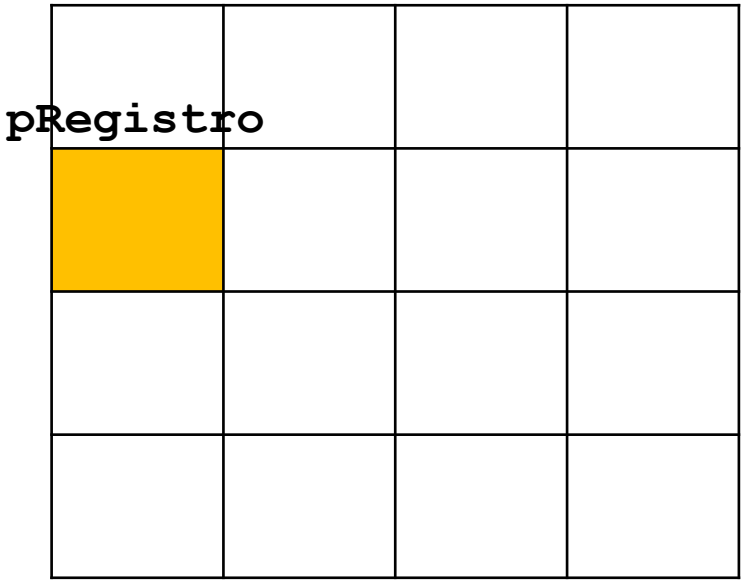
new: ejemplo (1/2)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ...
END.
```

Declaramos la variable estática (se reserva memoria en tiempo de compilación) puntero `pRegistro`



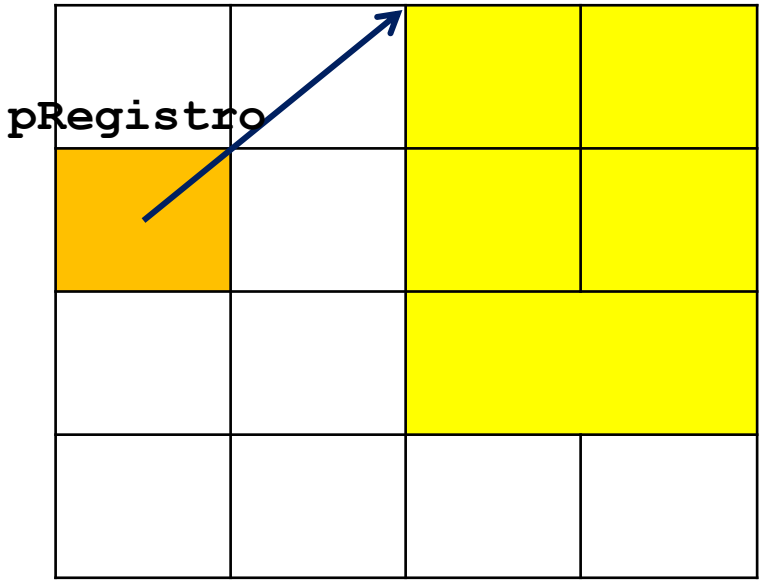
new: ejemplo (2/2)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ...
END.
```

new reserva memoria (para guardar un registro de tipo **tRegistro**) en tiempo de ejecución (dinámica) y el puntero que le pasamos (**pRegistro**) se deja apuntando a esa porción reservada .



Operación **`dispose`**

- **`dispose`**: procedimiento por el que se **libera memoria dinámica**
- Sintaxis:

```
dispose(variablePuntero);
```

- Semántica:
  - Libera el espacio de memoria apuntado por la variable puntero que se pasa como parámetro

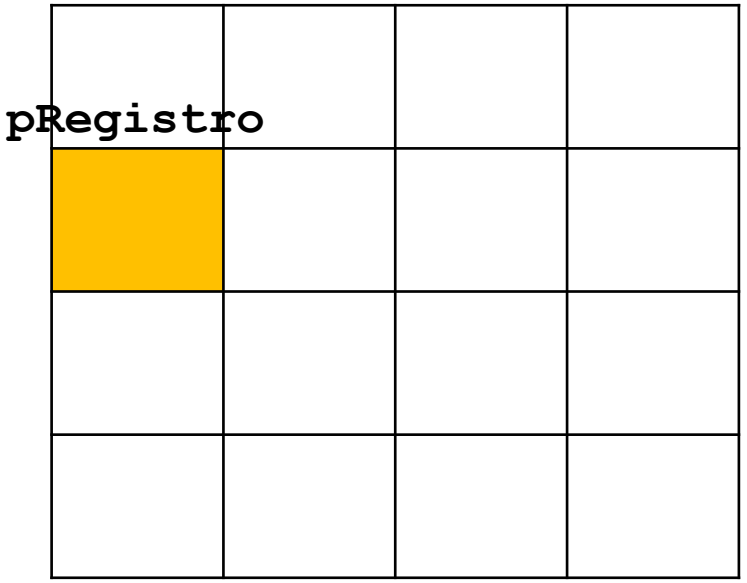
dispose: ejemplo (1/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  ...
END.
```

Declaramos la variable estática (se reserva memoria en tiempo de compilación) puntero `pRegistro`



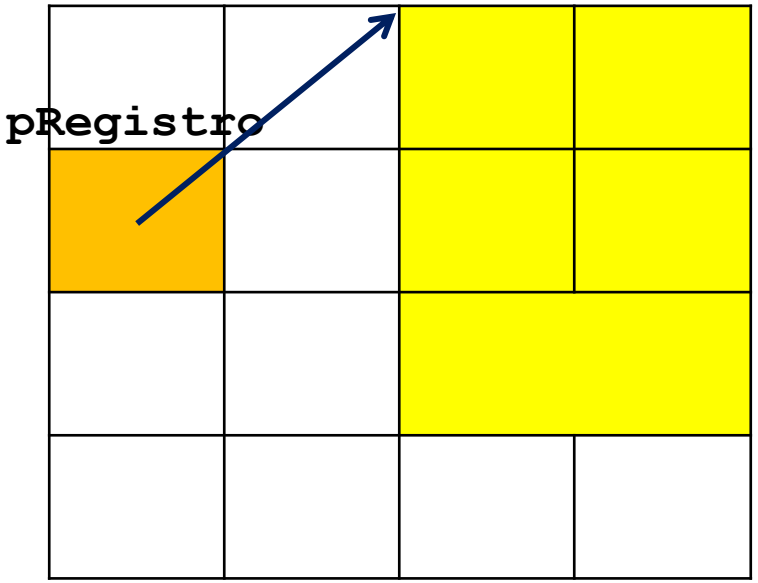
dispose: ejemplo (2/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  ...
END.
```

new reserva memoria (para guardar un registro de tipo **tRegistro**) en tiempo de ejecución (dinámica) y el puntero que le pasamos (**pRegistro**) se deja apuntando a esa porción reservada .



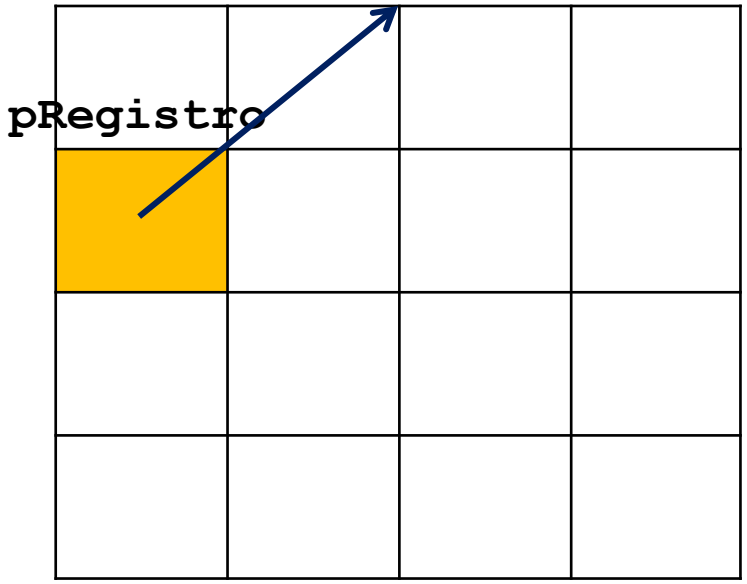
dispose: ejemplo (3/3)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  ...
END.
```

dispose libera la memoria apuntada por el puntero.



Puntero nulo: **`NIL`**

- **`NIL`** (Puntero nulo) es una constante de tipo puntero
- Un puntero **`NIL`**, indica que no apunta a ninguna posición de memoria

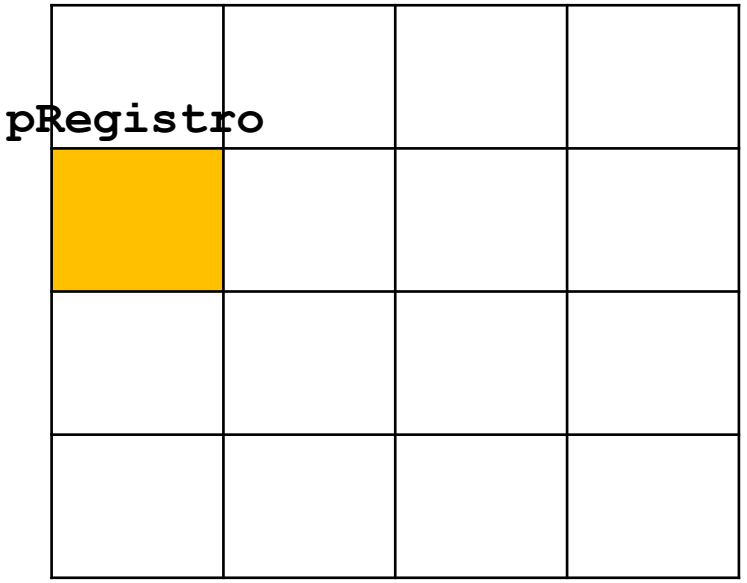
NIL: ejemplo (1/4)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  pRegistro := NIL;
  ...
END.
```

Declaramos la variable estática (se reserva memoria en tiempo de compilación) puntero `pRegistro`



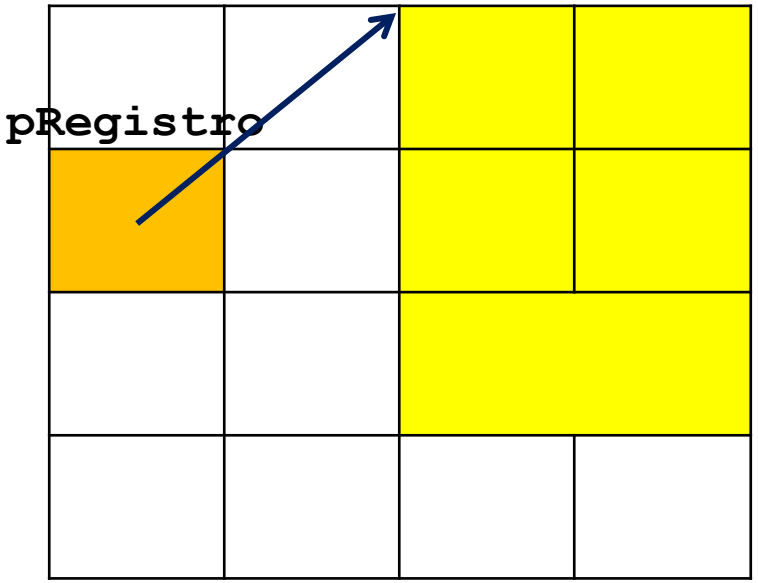
NIL: ejemplo (2/4)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  pRegistro := NIL;
  ...
END.
```

new reserva memoria (para guardar un registro de tipo **tRegistro**) en tiempo de ejecución (dinámica) y el puntero que le pasamos (**pRegistro**) se deja apuntando a esa porción reservada .



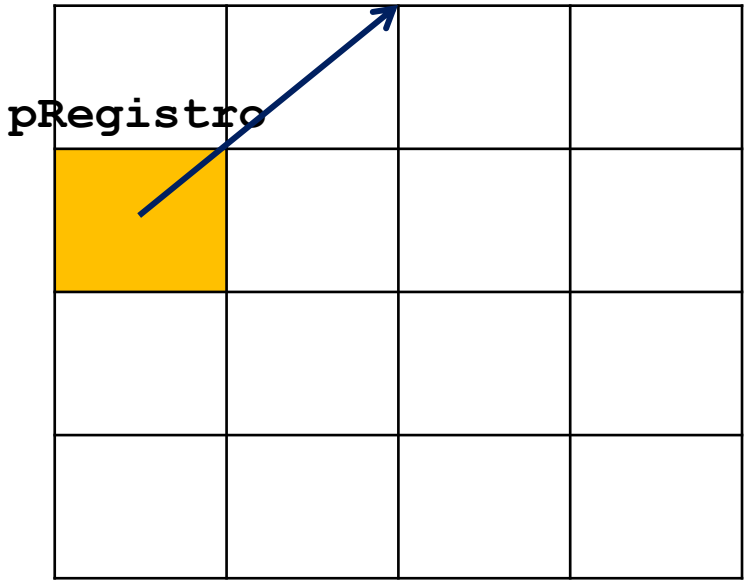
NIL: ejemplo (3/4)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  pRegistro := NIL;
  ...
END.
```

`dispose` libera la memoria apuntada por el puntero.



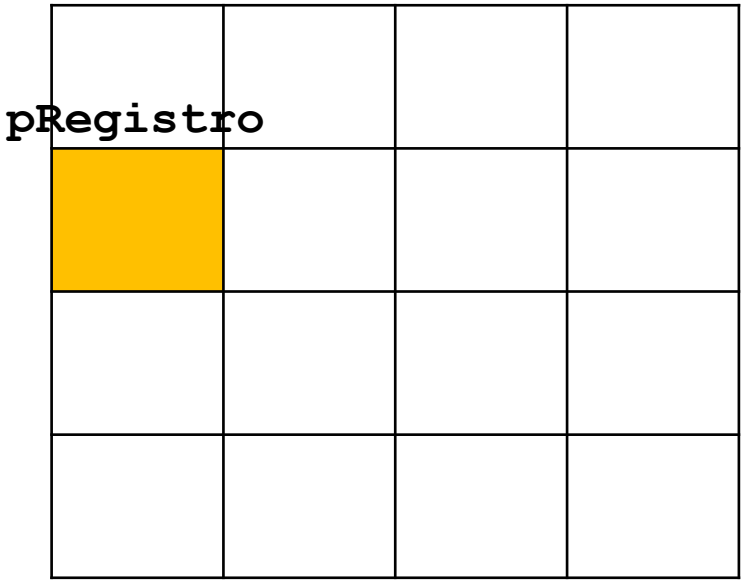
NIL: ejemplo (4/4)

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  dispose(pRegistro);
  pRegistro := NIL;
  ...
END.
```

asignar **NIL** al puntero, indica que no apunta a ninguna dirección de memoria.



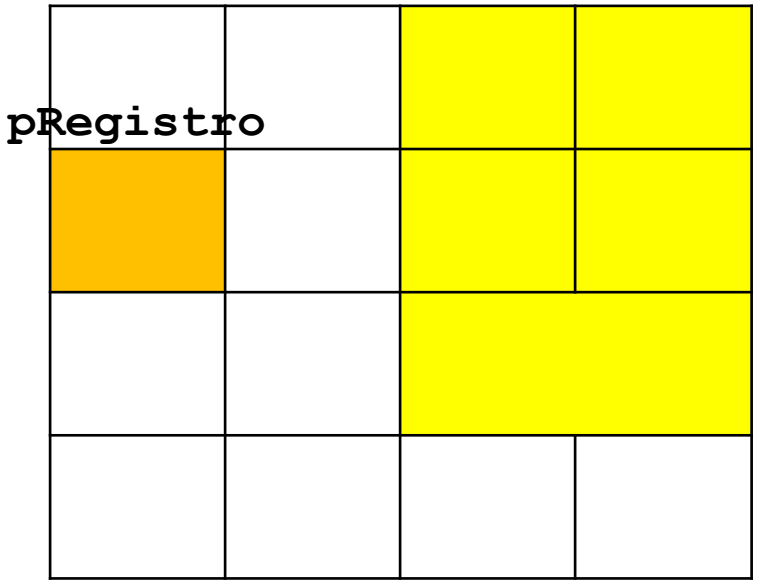
Efecto de omitir el **dispose**

```
TYPE
  tIniciales = string[3];
  tRegistro = RECORD
    iniciales: tIniciales;
    identificacion: Integer;
  END;
  tPtrRegistro = ^tRegistro;

VAR
  pRegistro: tPtrRegistro;

BEGIN
  ...
  new(pRegistro);
  ... {Utilización de pRegistro} ...
  pRegistro := NIL;
  ...
END.
```

Si, en el código anterior, se omite la liberación de memoria (mediante **dispose**), el resultado es que se mantiene reservada una porción de memoria innaccesible, ya que se ha desapuntado con la asignación de `NIL` a `pRegistro`.



Resumen y conclusiones

- **`variablePuntero ≠ variablePuntero^`**
- Es muy conveniente liberar el espacio de memoria dinámica mediante **`dispose`** cuando no se vaya a utilizar más
- Declarar una variable de tipo puntero no siempre es suficiente para poder utilizarla
- Para crear información dinámicamente se debe llamar al procedimiento **`new`**.
- Después de llamar a **`new`**, el valor al que apunta el puntero es indefinido. Para definirlo es necesario asignarle un valor.
- Se verán usos más interesantes de los punteros a lo largo de esta asignatura