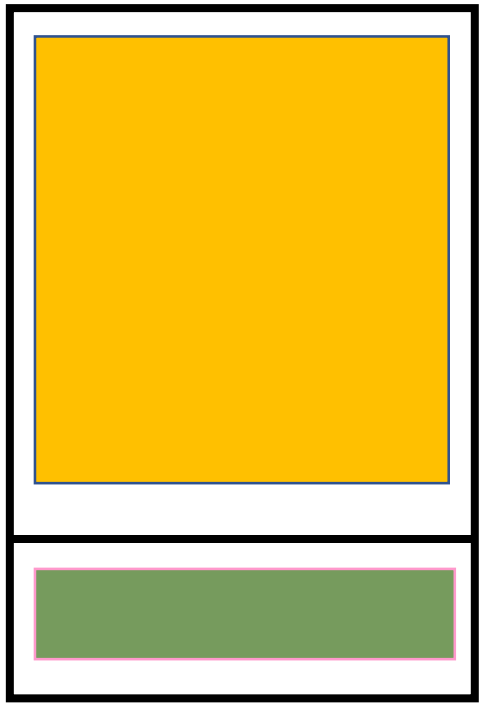
PUNTEROS



La MEMORIA ESTÁTICA. Acá se aloja los programas y las variables (del tipo integer, char, real, boolean, registro y arreglos). La capacidad de la memoria estática es limitada.

La alocación (reservación) estática reserva espacio en la memoria estática para las variables declaradas. Este espacio no se puede liberar en ningún momento durante la ejecución del programa.

La **MEMORIA DINÁMICA** o hip, trabaja con alocación dinámica, por lo cuál el espacio se puede liberar durante la ejecución del programa.

VARIABLES ESTÁTICAS

Son rígidas, ya que no permiten que las estructuras varíen su dimensión.

Las variables y tipos reservan memoria en su declaración. Se mantienen durante todo el programa. El lenguaje puede validar el espacio ocupado previo a la ejecución.

VARIABLES DINÁMICAS

Permiten modificar en ejecución la memoria utilizada

El espacio que ocupa depende de cada compilador y sistema operativo, pero los valores dados por la cátedra son:

En el teórico

Integer : 4B
Char : 1B
Real : 8B
Boolean : 1B
Registro : La suma de sus campos
Arreglos : DimF * tipo de elemento
String : Tantos B como su longitud lo indique + 1
Puntero : 4B

En el práctico

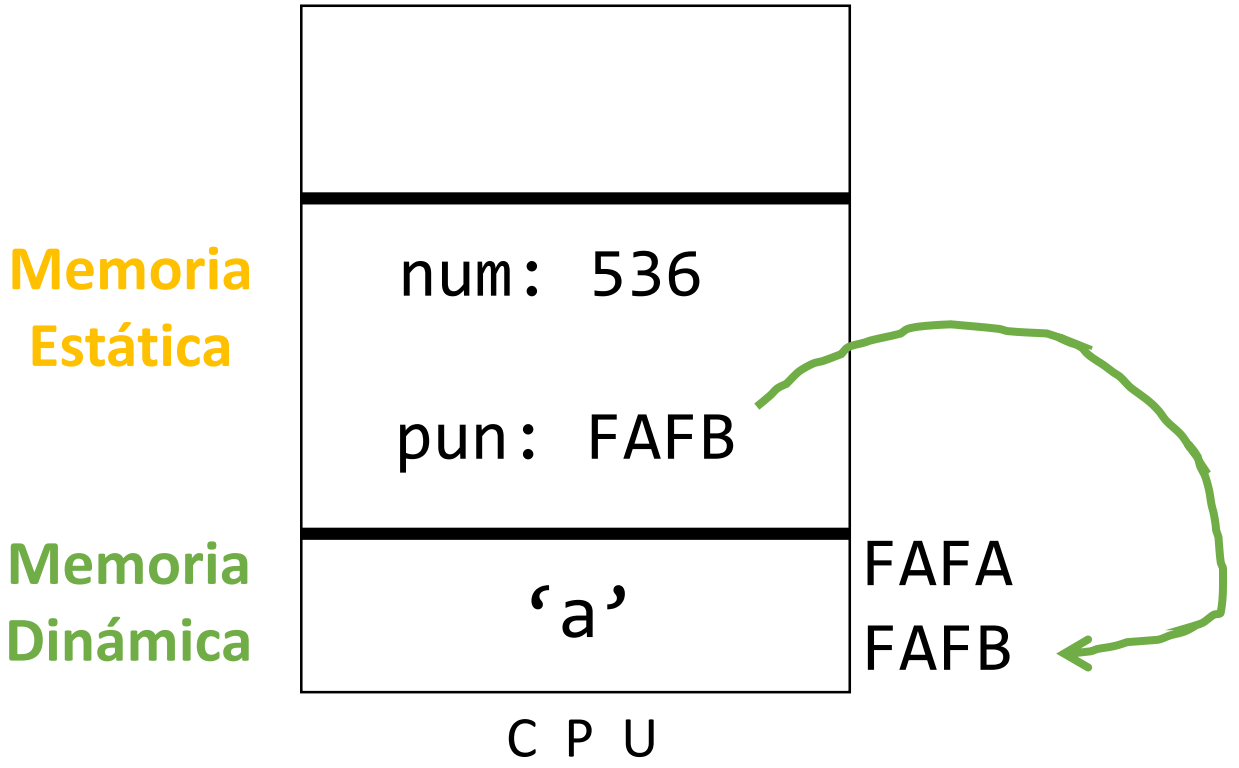
Integer : 2B
Char : 1B
Real : 4B
Boolean : 1B
Registro : La suma de sus campos
Arreglos : DimF * tipo de elemento
String : Tantos B como su longitud lo indique + 1
Puntero : 4B

SIZEOF (VARIABLE) es la que me permite conocer cuánto ocupa mi variable.

Un puntero es un tipo de variable usada para **almacenar la dirección en memoria de otra variable**, en lugar de un dato convencional. Mediante esta variable se accede a una dirección y, **en dicha dirección de memoria, se encuentra almacenado un valor.**

La variable de tipo puntero no tiene un dato, tiene una dirección que LLEVA a dónde está almacenado el dato.

- Es un tipo de **dato simple**. Contiene la dirección dónde se encuentra almacenado otro dato.
- Solamente apuntan a direcciones almacenadas en memoria dinámica (heap).
- Cada variable apunta únicamente a un tipo de dato.
- Ocupa una cantidad de memoria fija (4B) independientemente del tipo de dato al que apunta.
- **Puede reservar y liberar memoria durante la ejecución de un programa**



En este caso, la variable puntero (pun) contiene la dirección FAFB, que se almacena en la memoria dinámica. Esta dirección contiene el valor 'a'.

Es importante saber qué los datos referenciados (los que son apuntados por una variable del tipo puntero) no tienen memoria asignada. No existe inicialmente espacio reservado en memoria para ellos.

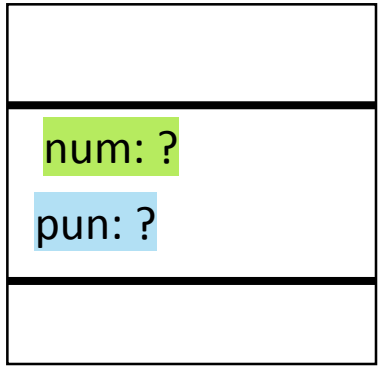
DECLARACIÓN

```
TYPE
    identificador= ^TipoVariableApuntada;
```

```
Program uno;
TYPE
  punteroAEntero = ^integer;

VAR
   pun:PunteroAEntero;
   num: integer;
```

Num es un entero y pun un puntero. Si bien pun apunta a un entero, son diferentes.



C P U

La variable puede ser de cualquier tipo de datos vistos hasta ahora.

Cuándo se ejecuta el programa, los punteros apuntan a direcciones en la memoria dinámica en la cuál asignan un tipo de dato y laburan con este. Luego, si se quiere, se libera este espacio de memoria.

OPERACIONES

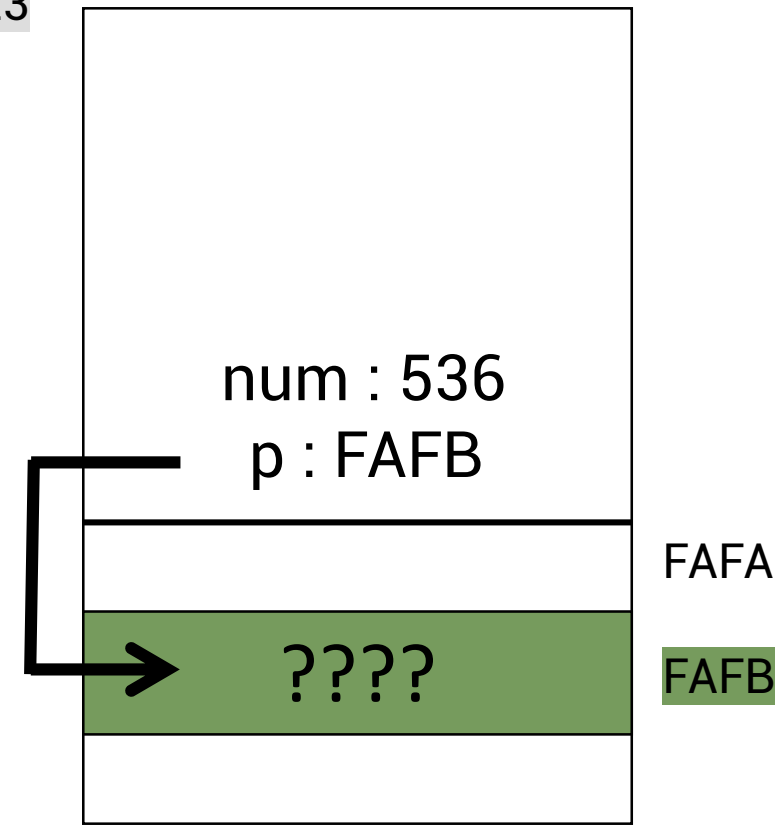
- Creación de una variable puntero // new (p) ;
- Destrucción de una variable puntero, // dispose (p)
- Liberación de una variable puntero, // p:= nil;
- Asignación entre variables punteros, // q := p; (ambas del tipo puntero)
- Asignación de un valor al contenido de una variable puntero // p^: = 123
- Comparación de una variable puntero

CREACIÓN DE UNA VARIABLE PUNTERO

```
new (p);
```

```
Program ejemplo;
Type
        puntero = ^integer;
Var
        num : integer;
        p : puntero;
Begin
        new (p);
        …
End.
```

Con new se asigna una dirección de memoria a la variable del tipo puntero. FAFB es la dirección de memoria que aún no tiene ningún contenido (pues no se le asignó ninguno) pero que ya está reservada por mi puntero p.

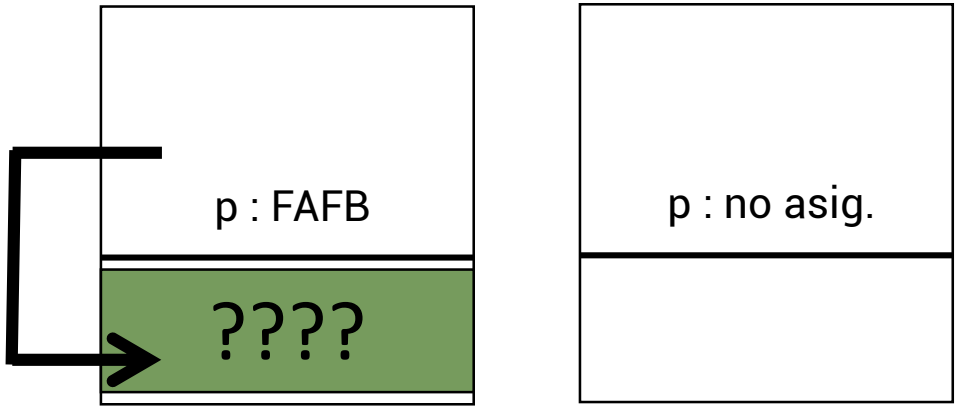


ELIMINACIÓN DE UNA VARIABLE PUNTERO

```
dispose (p);
```

```
Program ejemplo;
Type
        puntero = ^integer;
Var
        p : puntero;
Begin
        new (p);
        …
        dispose (p);
End.
```

Con dispose se libera el espacio en memoria, quedando borrada de la faz de la tierra y pascal la variable que allí se almacenaba y a la cuál nuestro puntero apuntaba. FAFB queda vacía

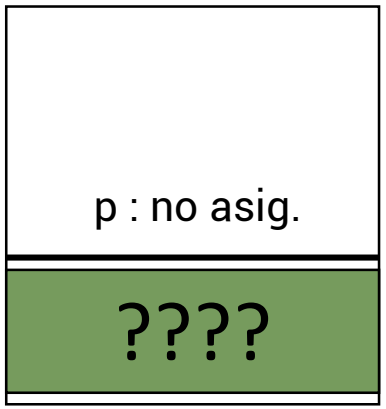


LIBERACIÓN DE UNA VARIABLE PUNTERO

```
p:=nil;
```

```
Program ejemplo;
Type
        punter = ^integer;
Var
        p : puntero;
Begin
        new (p);
        …
        p:=nil;
End;
```

La memoria queda ocupada (no se libera como en el dispose) pero no se puede acceder a ella. No queda valor asociado a la variable p. Al igual que el dispose, se deja de asignar una dirección de memoria a la variable puntero.



El dispose y el nil

Ambas tienen en **común** que se deja de asignar una dirección de memoria a la variable del tipo puntero, **se libera la conexión entre dicha variable y la posición de memoria** pero:

DISPOSE (p)

- Libera posición de memoria
- La memoria liberada puede utilizarse en otro momento del programa.

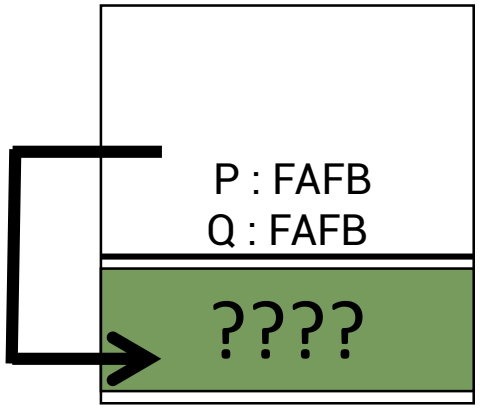
p := NIL

- La memoria sigue ocupada
- La memoria no se puede referenciar ni utilizar. Queda liberada una vez que termina el programa y se usa el *garbage colector.*

ASIGNACIÓN DE UNA VARIABLE PUNTERO

```
q := p;
```

```
Program ejemplo;
Type
        puntero = ^integer;
Var
        p : puntero;
        q : puntero;
Begin
        new (p);
        q := p;
End.
```



Para asignarse ambas tienen que ser del tipo puntero. La variable q va a tener la misma dirección y contenido que p. Se comparte entre ambas variables, lo cuál lo relaciona muchísimo con los parámetros por referencia.

Algo muy curioso es que si yo ahora hago

```
Begin
        new (p);
        q := p;
        p := nil;
End.
```

La memoria queda ocupada, p no puede acceder a la misma (porque se 'borra el enlace') pero si puede acceder q. En cambio:

```
Begin
        new (p);
        q := p;
        dipose (p);
End.
```

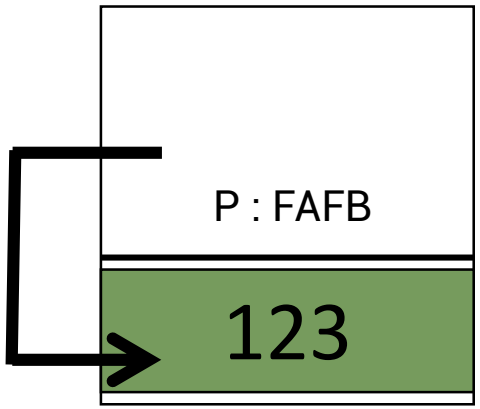
Queda liberado por completo el espacio de memoria, así que tampoco q puede acceder a lo que había allí, por el espacio ya no está.

DARLE CONTENIDO A UNA VARIABLE PUNTERO

```
p^ := 123;
```

```
Program ejemplo;
Type
        puntero = ^integer;
Var
        p : puntero;
Begin
        new (p);
        p^ := 123;
End.
```

La variable puntero debe tener una dirección asignada para poder darle contenido. Las operaciones permitidas para el contenido depende del tipo que se apunta.



HERRAMIENTAS ÚTILES

- **if (p = nil) then …** Esto compara si el puntero p no tiene dirección asignada.

- **if (p = q) then …..** Esto compara si los punteros p y q apuntan a la misma dirección. AMBOS TIENEN QUE SER PUNTEROS.

- **If (p^=q^) then …** Esto compara si los punteros p y q tienen el mismo contenido. AMBOS TIENEN QUE SER PUNTEROS.

COSAS QUE NO SE PUEDEN HACER

- No se puede hacer **read (p),** si p es una variable de tipo puntero.

- No se puede hacer **write (p),** si p es una variable de tipo puntero.

- Se puede asignar una dirección a un puntero manualmente, pero NO SE DEBE. Porque podes hacer cagadas ☺ NO SE DEBE HACER **(p := ABCD).**

- No se puede realizar **operaciones de mayor o menor** entre las direcciones de los punteros **(p>q)** y si ambos son el mismo tipo de puntero, no tendría sentido ya que ambos apuntarían a la misma dirección.