Explicación práctica N°1: Máximos y mínimos en Pascal

- **Ejercicio 1**: Realizar un programa que lea números enteros desde el teclado. La lectura finaliza cuando se ingrese el número 0, **el cual no debe procesarse**. **Informar el número máximo**.
- **Ejercicio 2** : Se leen las alturas de 20 jugadores de básquet junto con su DNI. Informarlos DNI de los 2 jugadores más altos.

```
Program ejercicio1 ;
Var
    Nro , Max : integer ;
Begin
    Max := -1 ;
    {Leo un número}
    Read (Nro) ;
    While (Nro <> 0) do begin
        {Actualizo el máximo}
        If (Nro > Max) then
            Max := Nro ;
        {Leo otro número}
        Read (nro) ;
    End;
    Writeln ('El número más alto fue: ', Max) ;
End.
```

```
Program ejercicio2 ;
Var
    altura , max1 , max2 : real ;
    dni , dnimax1 , dnimax2 , i : integer ;
Begin
    max1 := -1 ; max2 := -1 ;
    For i := 1 to 20 do begin
        read (altura) ; read (dni) ;
        If (altura > max1) then begin
            max2 := max1 ;
            dnimax2 := dnimax1 ;
            max1 := altura ;
            dnimax1 := dni ;
        End;
        Else
            If (altura > max2) then begin
                max2 := altura ;
                dnimax2 := dni ;
            end;
    end;
    Writeln ('DNI 1er jugador más alto:' , dnimax1) ;
    Writeln ('DNI 2do jugador más alto:' , dnimax2) ;
End.
```

Explicación práctica N°2: Estructuras de control. Manejo de caracteres (Parte 1/2)

- **Ejercicio 1**: Se lee una secuencia de caracteres terminada en '.'. Informar la cantidad de caracteres 'a' leídos.
- **Ejercicio 2** : Se leen una secuencia de caracteres terminados en '.'. La secuencia está dividida en palabras, separadas por uno o más blancos. Informar la cantidad de palabras.

```
Program ejercicio1 ;
Var
    cant: integer ;
    car : char ;
Begin
    cant := 0 ;
    Read (car) ;
    While (car <> '.') do begin
        If (car = 'a') then
            cant := cant +1 ;
        Read (car) ;
    End;
    Writeln ('Cantidad de letras a leídas: ', cant) ;
End.
```

```
Program ejercicio2 ;
Var
    cantPal : integer ;
    car : char ;
Begin
    cantPal := 0 ;
    read (car) ;
    while (car = ' ' ) do                    {descarto blancos}
        read (car) ;
    while (car <> '.' ) do begin      {cuento la palabra que empieza}
        cantPal := cantPal + 1 ;
        while (car <> '.' ) and (car <> ' ') do {Leo el resto de la palabra}
            read  (car) ;
        while (car = ' ' ) do                {descarto blancos}
            read  (car) ;
    end;
    Writeln ('La cantidad de palabras es:' , cantPal) ;
End.
```

- **Ejercicio 2.1** : Contando las palabras que empiezan con M.
- **Ejercicio 2.1.1**: Contando las palabras que empiezan con M y terminan con S.

```
Program palabra1 ;
Var
    cantPal : integer ;      {contador de palabras}
    car : char ;
Begin
    cantPal := 0 ;
    Read (car) ;
    {Descarto blancos}
    While (car = ' ' ) do
        Read (car) ;
    While (car <> '.') do begin
        {Cuento la palabrta solo si empieza con M}
        If (car = M) then
            cantPal := cantPal + 1 ;
        {Leo el resto de la palabra}
        While (car <> '.' ) and (car <> ' ' ) do
            Read (car) ;
        {Descarto blancos}
        While (car = ' ') do
            Read (car) ;
    End;
    Writeln ('La cantidad de palabras es: ', cantPal) ;
End.
```

```
Program palabra2 ;
Var
    cantPal : integer ;      {contador de palabras}
    car, ant : char ;
Begin
    cantPal := 0 ;
    Read (car) ;
    {Descarto blancos}
    While (car = ' ' ) do
        read (car) ;
    While (car <> '.') do begin ;
        If (car = M) then begin
            {Leo el resto de la palabra}
            While (car <> '.' ) and (car <> ' ' ) do begin
                ant := car ;
                read (car) ;
            end;
            If (ant = 'S') then {Si la última letra fue una S cuento la palabra}
                cantPal := cantPal +1 ;
        end;
        Else
            While (car <> '.' ) and (car <> ' ' ) do
                read (car) ;
            While (car = ' ') do {descarto blancos}
                read (car) ;
    end;
    Writeln ('La cantidad de palabras es: ', cantPal) ;
End.
```

Explicación práctica N°2: Estructuras de control. Números (Parte 2/2)

- **Descomponer un número**:
  - ✓ Con **MOD** (10) obtengo el último dígito.
  - ✓ Con **DIV** (10) achico el número un dígito (repetir hasta 0)
- **Averiguar si un número es par**:

✓ Si [MOD 2] da resto 0, es par.

- **Ejercicio 2**: Se leen desde el teclado 5 números enteros. Informar la cant de dígitos pares que posee cada nro.

```
Program descompone ;
Var
    i, número, n, dígito, cant: integer ;
Begin
    For i := 1 to 5 do begin
        Read (numero) ;
        cant := 0 ;
        n := numero ;
        While (n <> 0) do begin
            digito := n MOD 10 ;
            if ((digito MOD 2) = 0) then
                cant := cant +1 ;
            n := n DIV 10 ;
        end;
        Writeln (numero 'tiene' , cant, 'digitos pares.') ;
End.
```

Explicación práctica N°3: Modularización. Procedimientos y funciones

- **Procedimientos:**

```
Program NombrePrograma ;
Type
    {Declaraciones de tipos de datos}
Procedure nombre (lista de parámetros) ;
Var
    {Variables locales al procedimiento}
Begin
    {Cuerpo del procedimiento}
End;

Var
    {Variables a usar en el programa principal}
Begin
    {Acciones del programa principal}
    Nombre (param actuales)  →Por posición
End.
```

✓ **Variables locales al procedimiento**
- Solo conocidas por el módulo
- Se crean cuando se invoca al módulo
- Desaparecen al llegar al end del módulo

✓ **Variables a usar en el programa principal**
- Accesibles solo por el programa principal
- Se crean al ejecutarse el programa principal
- Desaparecen al llegar al end del programa ppal

Lista de parámetros→Parámetros formales. Para cada parámetro:
- Tipo de pasaje: por valor o por referencia (VAR)
- Nombre del parámetro
- Tipo de dato

→**Parámetros por valor** (similar a los parámetros de entrada (E))

- El módulo necesita recibir información para realizar su tarea
- El llamador no ve modificaciones en el parámetro actual

[*Mecanismo*]: Se realiza una copia del valor del parámetro actual en otra posición de memoria correspondiente al parámetro formal (la copia se destruye cuando termina de ejecutar el módulo)
Dentro del módulo se puede modificar el valor del parámetro, pero el cambio no será visto por el llamador

→**Parámetro por referencia** (Similar a los parámetros de entrada/salida (ES))

- El módulo necesita opcionalmente recibir un dato, procesarlo, y devolverlo al llamador.
- Cuando se modifica el parámetro formal, la modificación es vista por el módulo llamador.

[*Mecanismo*]: El parámetro formal recibe la dirección de memoria donde se encuentra la variable que se pasó como parámetro actual.
*Parámetro actual y formal "comparten" el mismo espacio de memoria.

Cuando hay mas de un parámetro:
Procedure actualizarMax (cantAct : integer ; nombreAct :string ; var max :integer ; var nomMax : string);
→Los parámetros se leen por posición.

```
Program paramValor ;
Procedure uno (a : integer) ;
Var
    Total : integer ;
Begin
    Total := 3 ;
    Total := total + a ;
    a := a + 1 ;
    Writeln ('El valor de total es:' , total) ;
    Writeln ('El valor de a es:' , a) ;
End;
Var
    X : integer ;
Begin
    X := 30 ;
    Uno(x)  ;
    Writeln ('El valor de x es' , x) ;
    Readln
End.
```

```
Program paramReferencia ;
Procedure uno (var a : integer) ;
Var
    Total : integer ;
Begin
    Total := 3 ;
    Total := total + a ;
    a := a + 1 ;
    Writeln ('El valor de total es:' , total) ;
    Writeln ('El valor de a es:' , a) ;
End;
Var
    X : integer ;
Begin
    X := 30 ;
    Uno(x)  ;
    Writeln ('El valor de x es' , x) ;
    Readln
End.
```

- **Funciones:**

```
Program NombrePrograma
Type
    {Declaraciones de tipos de datos}
Function nombre (lista de parámetros) : tipo[1] ;
Var
    {Variables locales al procedimiento}
Begin
    {0,1 o más sentencias}
    Nombre := valor[1] ;
End;
Var
    {Variables a usar en el programa principal}
Begin
    {Acciones del programa principal}[2]
End.
```

Parámetros formales→Solo parámetros por valor. Para cada parámetro:

- Nombre del parámetro
- Tipo de dato

[1]Las funciones retornan un dato simple

[2]La invocación a una función puede hacerse de distintas formas

[Invocación]: **function** cuadrado (a : real) : real[1] ;

- **Asignando el valor a una variable**
  →cuad1 **:=** cuadrado (x)[1]
  →cuad2 **:=** cuadrado (y)[1]

  [1]Tienen que ser del mismo tipo
- **Dentro de un write** (si el valor retornado se puede imprimir)
  →**Writeln** (cuadrado (x), cuadrado (y))
- **Dentro de una expresión de una condición:**

```
Function verificar (carácter : char) : boolean ;
Begin
    Verificar := (carácter >= '0') and (carácter <= '9') ;
End;
…
If (verificar (carac)) then
…
End.
```

Explicación práctica N°4: Registros.

- **Declaración:**

```
nombreTipoReg = record
        campo1 : tipo-campo1 ;
        campo2 : tipo-campo2 ;
end;
```

- **Declaración de variable**: miVariable : nombreTipoReg ;
- **Acceso a campo**: miVariable.campo1

Operaciones:

- **Leer** (campo a campo)
- **Imprimir** (campo a campo)
- **Comparar** (campo a campo)
- **Asignar** (:=)

```
Program Secretaria ;
Type
    Sitio = record
        Nombre : string ;
        Prov : string ;
        cantAct : integer ;
        cantVis : integer ;
    end ;
var
    sitioTur : sitio ;
    cant, max : integer ;
    nomMax : string ;
procedure LeerRegistro (Var S : sitio) ;
begin
    with S do begin
        read (nombre) ;
        if (nombre <> 'fin') then begin
            read (prov) ;
            read (cantAct) ;
            read (cantVis) ;
        end ;
    end ;
end;
begin      {Cuerpo del programa principal}
    cant := 0 ;
    nomMax := ' ' ;
    max := -1 ;
    LeerRegistro (sitioTur) ;  {Se lee el registro}
    While (sitioTur.nombre <> 'fin') do begin
        If (sitioTur.CantAct > max) then begin
            Max := sitioTur.CantAct ;
            nomMax := SitioTur.Nombre ;
        end;
        if (sitioTur.CantVis > 2000) then
            cant := cant + 1 ;
        {Siguiente registro}
        LeerRegistro (SitioTur) ;
    End;
    Write ('Sitio con más actividades:' , nomMax) ;
    Write ('Sitio con más de 2000 visitas:' , cant) ;
End
```

Explicación práctica N°5: Arreglos. Aspectos básicos

- **Dimensión física:** La asignada al vector en su declaración. Elementos máximos que tendrá el vector (no varía)(10)
- **Dimensión lógica:** Posición del último elemento que se cargó en el vector. Debe controlarla el programador a medida que agrega/quita elementos (varía) (5)

→Declaración de tipo VECTOR

```
NombreTipo=Array [Rango][1] of tipoElem[2] ;
```

[1]El rango debe ser no nulo de un tipo ordinal ('a'..'z' – 0..9)
[2]El tipoElem debe ser un tipo permitido y/o previamente definido

```
Type
    Vector=array[1..10] of integer
Var
    v : vector ;
    dl : integer ;
```

V	10	14	19	25	33					
	1	2	3	4	5	6	7	8	9	10

- ✓ **Acceso directo**: Se puede acceder a un elemento mediante la especificación de la posición donde se encuentra en el vector
- ✓ Acceso al elemento en posición 5: V[5]
- ✓ **Imprimir un arreglo**:

```
…
 For i := to dl do
      Write (V[i]) ;       {Imprime todo el arreglo}
…
```

→Se pueden declarar vectores de: integer, real, char, boolean, subrango, string, registro, vectores.

Explicación práctica N°6: Punteros

- ✓ **Variables estáticas**: Se conoce el tamaño máximo que van a ocupar en memoria
  →Se reserva memoria en el momento que comienza la ejecución del módulo que la declara

Char	1 byte
Integer	4 bytes
Real	8 bytes
Boolean	1 byte
String	Cantidad de caracteres + 1
Registro	La suma de lo que ocupa c/campo
Puntero	4 bytes
Vector	(dim. Física) x (tam de dato almacenado en bytes)

- **Punteros**: Permiten reservar y liberar memoria en tiempo de ejecución, para crear estructuras de datos dinámicas cuya cantidad de elementos varía (cuando no se conoce la cantidad de elementos a priori).
- **Variable puntero**: almacena la dirección en memoria de otra variable (llamada variable dinámica)

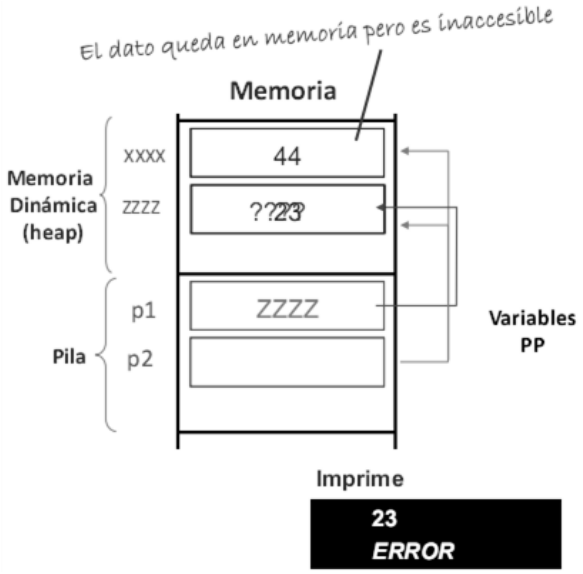
Memoria	
Variables dinámicas	Datos apuntados por punteros
Variables estáticas	Char / integer / string / registro boolean / arreglo / **puntero**

Uso de los punteros:

Declaración (Ejemplo)	Valores posibles de un puntero
TYPE PtroEntero = ^ integer **VAR** Pun , otropun : PtroEntero	▪ Nil ▪ Dirección de memoria obtenida
Operaciones ▪ **New (Pun)** adquiere la memoria necesaria para una *variable dinámica*. El valor de pun es la dirección de memoria en donde se guarda el dato del tipo apuntado ▪ **Dispose (pun)** libera la memoria adquirida mediante el *new().* El puntero queda con valor indefinido ▪ **Asignación** ✓ Entre punteros de igual tipo. *Ej: Pun := otropun* ; ✓ Asignación de Nil. *Ej: Pun := NIL* ; ▪ **Comparación**: ✓ Entre punteros de igual tipo. *Ej: if (pun = otropun) then…* ✓ Comparación con Nil. *Ej: if (pun <> NIL) then…*	
Acceder al dato apuntado por el puntero *pun^*	

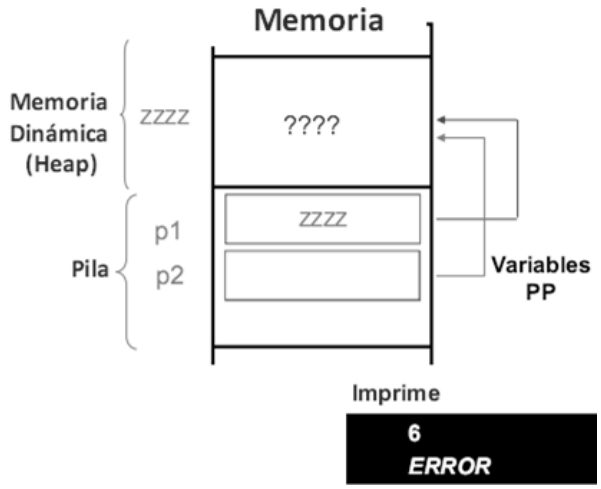
Ejemplo 1:

```
Program ejemplo1 ;
Type
    Ptro = ^integer ;
Var
    P1 ,P2 : Ptro ;
Begin
    New (P1) ;
    P1^ := 23 ;
    New (p2) ;
    P2^ := 44 ;
    P2 := P1 ;
    Write ( P2^) ;
    Dispose (P2) ;
    Write (P1^) ;
End.
```



Ejemplo 2:

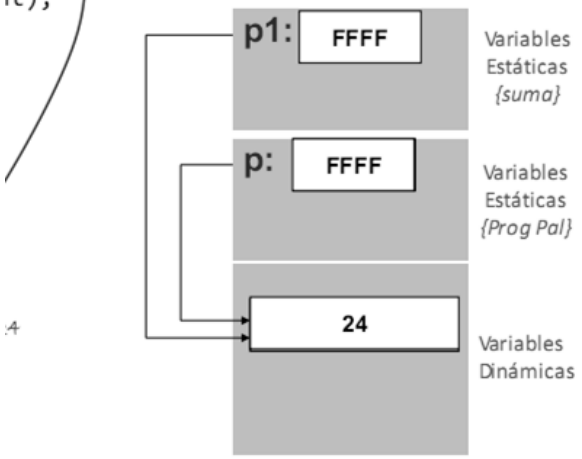
```
Program ejemplo2 ;
Type
    Casa = record
        Met_cua : real ;
        Cant_hab : integer ;
    End;
    Punt_casa = ^casa ;
Var
    P1 ,P2 :Punt_casa ;
Begin
    New (P1) ;
    P1^.met_cua := 125.50 ;
    P1^.cant_hab := 5 ;
    P2 := P1 ;
    P2^.cant_hab := 6 ;
    Write ( P1^cant_hab) ;
    Dispose (P2) ;
    Write (P1^.met_cua) ;
End.
```



Punteros como parámetros:

```
Program ejemplo ;
Type
     Punt = ^integer ;
Procedure suma (p1 : punt) ;
Begin
     P1^ + 1 ;
End ;
Var
     P : punt ;
Begin
     New (p) ;
     P^ := 23 ;
     Suma (p) ;
     Write (p^) ;        {Imprime 24}
End.
```

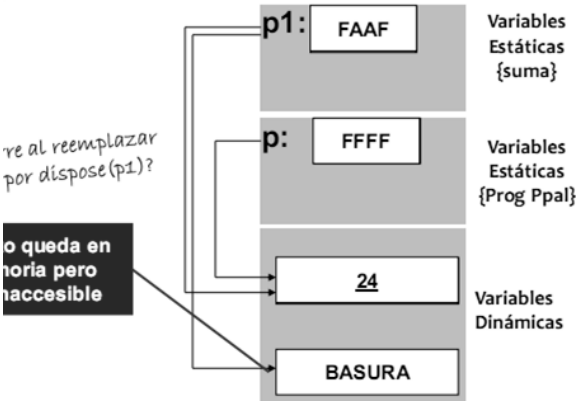
p1 recibe una <u>copia</u> de la dirección de **p**. Si modifico el <u>dato apuntado</u> por **p1**, el llamador verá el cambio.




```
Program ejemplo ;
Type
     Punt = ^integer ;
Procedure suma (p1 : punt) ;
Begin
     P1^ := p1^ + 1 ;
     New (p1) ;
End ;
Var
     P : punt ;
Begin
     New (p) ;
     P^ := 23 ;
     Suma (p) ;
     Write (p^) ;        {Imprime 24}
End.
```

p1 recibe una recibe una <u>copia</u> de la dirección de **p**. Si modifico la <u>dirección</u> **p1**, **p** en el programa ppal tiene la misma dirección que tenía antes de la llamada.

¿Qué imprime si modifico suma?




```
Program ejemplo ;
Type
     Punt = ^integer ;
Procedure suma (VAR p1 : punt) ;
Begin
     P1^ := p1^ + 1 ;
     New (p1) ;
     P1^ := 44 ;
End ;
Var
     P : punt ;
Begin
     New (p) ;
     P^ := 23 ;
     Suma (p) ;
     Write (p^) ;        {Imprime 44}
End.
```

p1 recibe la <u>referencia</u> de **p**. Si modifico la <u>dirección</u> **p1**, estoy modificando la dirección de **p** en el programa ppal.

¿Qué imprime si modifico suma?