TEORIA 6

TEMAS de la CLASE

1. Modularización

2. Mecanismos de comunicación
   - Variables globales
   - Parámetros

Modularización – Comunicación de Datos

En la clase anterior hemos visto una manera muy simple de compartir datos entre los módulos y el programa, que es a través del uso de variables globales. Entonces el programa se puede comunicar con los módulos utilizando estas variables.

LOS DATOS PUEDEN COMUNICARSE ENTRE MODULOS Y PROGRAMA

↓

VARIABLES GLOBALES

```
Program dos;
Var
 a, b: integer;

procedure calculo(parámetros formales);
  var
      x: integer;
  Begin
      x:= 9; a:= 100;
      write (x);
  End;

 Var
  h: char;
Begin
  a:= 80;
  b:= a * 2;
  h:= 'A';
  calculo(parámetros actuales);
End.
```

Variables globales

Parámetros

Variables Locales del módulo

Variable Local del programa

Comunicación mediante Variables globales

El uso de variables globales para la comunicación presenta desventajas:

- Demasiadas variables en la sección de declaración

- Puede ocurrir conflicto entre los nombres de las variables utilizadas por diferentes programadores

- No se especifica que tipo de comunicación existe entre los módulos.

- Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo datos de alguna variable que luego deberá utilizar otro módulo.

Se recomienda evitar el uso de variables globales

Se recomienda una solución que combina:

OCULTAMIENTO DE DATOS Y PARAMETROS

- El **ocultamiento de datos** significa que los datos exclusivos de un módulo o programa **NO** deben ser "visibles" o utilizables por los demás módulos.

- El **uso de parámetros** significa que los datos compartidos se deben especificar como parámetros que se trasmiten entre módulos.

Variables locales del módulo

Variables locales del programa

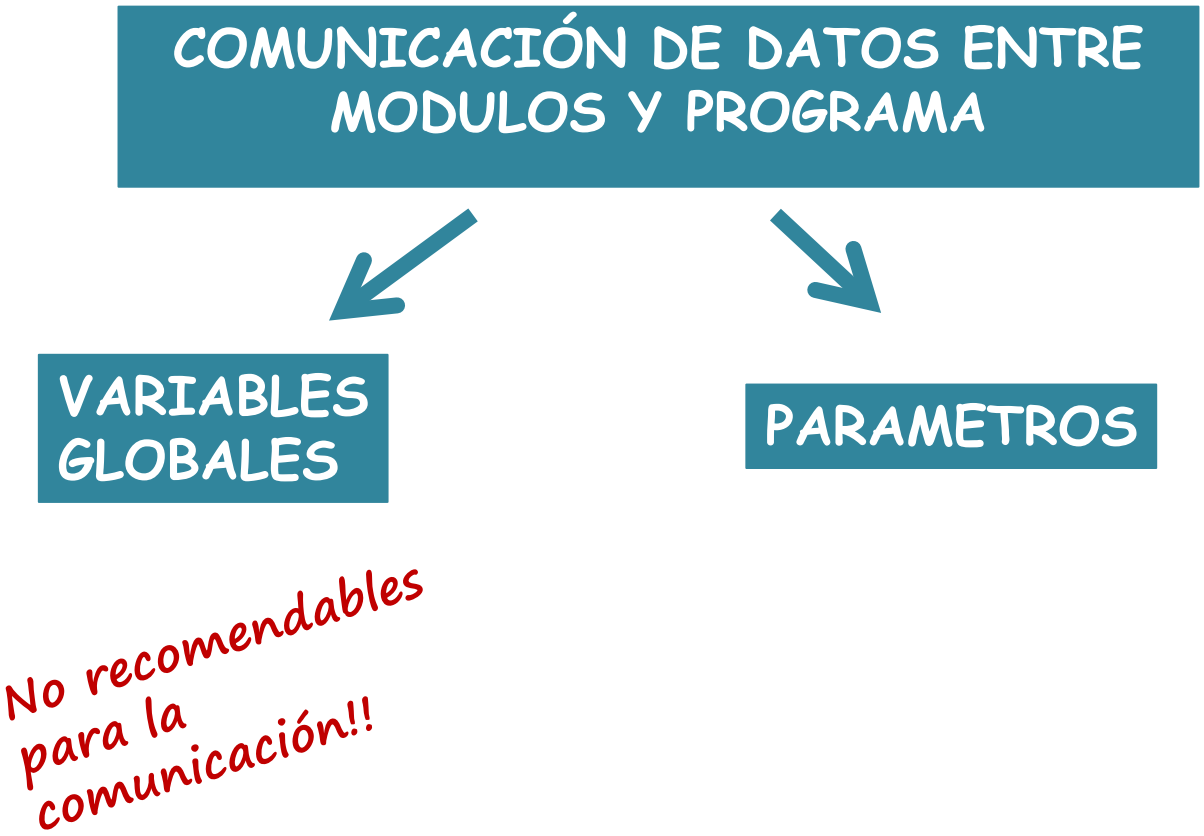
Los datos propios del módulo se declararan locales al módulo

Los datos propios del programa se declararan locales al programa

Los datos compartidos se declararán como parámetros.

No los usaremos para la comunicación!!

Modularización – Comunicación de Datos



Comunicación mediante Parámetros

En Pascal, la comunicación entre módulos usando parámetros puede ser por valor o por referencia.

Por valor

- Un parámetro por valor es un dato de entrada que significa que el módulo recibe una copia de un valor proveniente de otro módulo o del programa principal.

  Con este dato el módulo puede realizar operaciones y/o cálculos, pero fuera del módulo ese dato NO reflejará cambios.

Comunicación mediante Parámetros

En Pascal, la comunicación entre módulos usando parámetros puede ser por valor o por referencia.

Por referencia

- Un parámetro por referencia es un dato que contiene la dirección de memoria donde se encuentra la información compartida con otro módulo o programa que lo invoca.

El módulo que recibe este parámetro puede operar con la información que se encuentra en la dirección de memoria compartida y las modificaciones que se produzcan se reflejarán en los demás módulos que conocen esa dirección de memoria compartida.

¿Cómo se distingue un parámetro por referencia de un parámetro por valor?

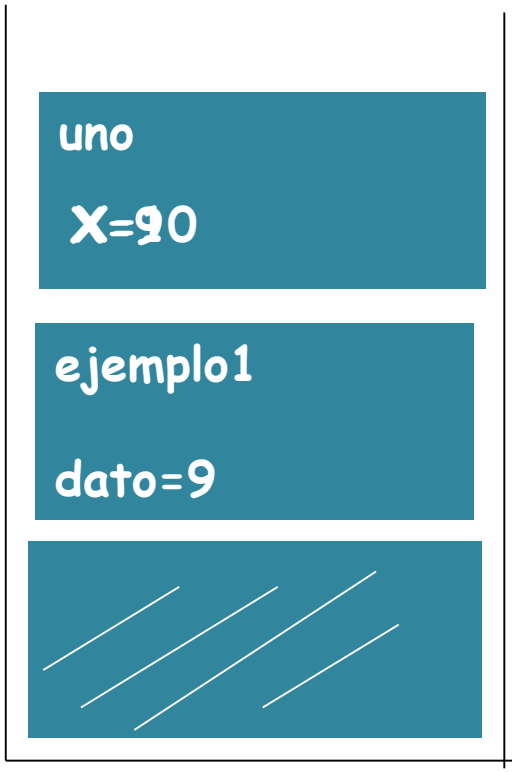
Comunicación mediante Parámetros por valor

```
Program ejemplo1;

Procedure uno (x:integer);
  Begin
    x:= x+1;
    write (x);
  End;

var dato: integer;
Begin
  dato:=9;
  uno (dato);
  write (dato);
End.
```

Qué valores imprime?



Memoria

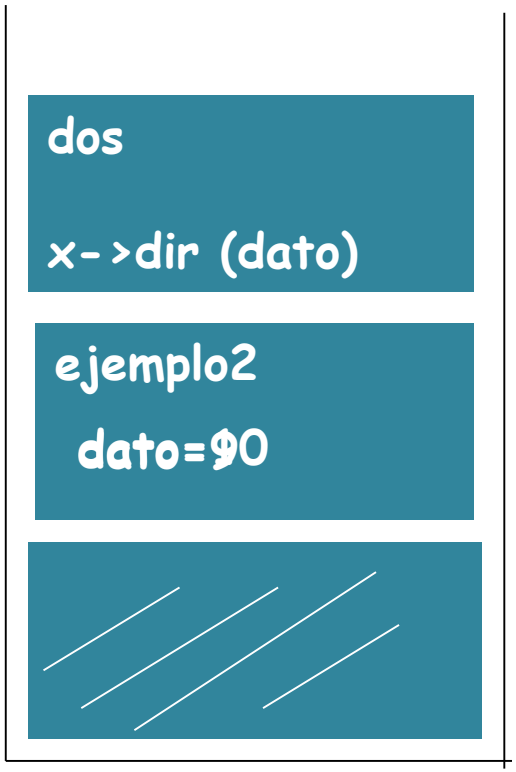
Comunicación mediante Parámetros por referencia

```
Program ejemplo2;

Procedure dos (var x:integer);
  Begin
    x:= x+1;
    write (x);
  End;

var dato: integer;
Begin
  dato:=9;
  dos (dato);
  write (dato);
End.
```

Qué valores imprime?



Memoria

Consideraciones generales

Cuando se invoca a un procedimiento se deben tener cuenta:

- Su invocación
- La relación entre los parámetros formales y actuales

Consideraciones generales: INVOCACION

```
Program ejemplo3;
  Procedure Calcular (x, y: integer;
            var suma, prod: integer);
  begin
    suma:= x + y;
    prod:= x * y;
  end;

Var  val1, val2, s, p: integer;

Begin
  Sumar (6, 17, s, p);

  val1:= 10; val2:= 5;
  Sumar (val1,val2,s, p);

  Sumar (23, val2, 14, p);
End.
```

OJO!!!

¿Cuáles son las invocaciones Válidas del programa ?

¿Qué tengo que tener en cuenta?

¿Parámetro por valor?

¿Parámetro por referencia?

El número y tipo de los parámetros actuales que se utilizan en la invocación a un módulo deben coincidir con el número y tipo de parámetros formales del encabezamiento del módulo.

```
Program Ejemplo4;
  procedure Imprimir (var a:integer; b: integer; c: integer)
    begin
      writeln (a, b, c);
      a := 10;
      c := a + b;
      b := b * 5;
      write (a, b, c);
    end;
 var x, y, z: integer;
begin
  y := 6;
  z := 5;
  writeln (x, y, z);
  imprimir (x, y, z);
  writeln (x, y, z);
end.
```

Analicemos ¿cómo se modifican las variables?

¿Qué se imprime?

Observar que la variable local x no está inicializada

¿Qué imprime en cada sentencia write?

```
program ejemplo5;

procedure cuenta(var b: integer; var a: integer);

  procedure calculo (var b: integer; a: integer);
   begin
     b := a * 2 + 4;
     writeln(a, b);
   end;

var c: integer;
begin
 b := a * 2;
 calculo(a,b);
 c:= 0;
 writeln(a, b, c);
end;

Var a, b,c: integer;
begin
 a:= 5;
 b:= 20 div 10;
 writeln(a, b, c);
 cuenta(c, a);
 writeln(a, b, c);
end.
```

Observar la declaración de la variable local c

¿Qué imprime en cada sentencia write?

```
Program Ejemplo6;
var a: integer;

  procedure Imprimir (var b: integer; c: integer);
   begin
      writeln (a, b, c);
      a := a + 5;
      c := a + b;
      b := b * 5;
      write (a, b, c);
    end;
var b, c: integer;
begin
   a := 20;
   b := 6;
   writeln (a, b, c);
   imprimir (b, a);
   writeln (a, b, c);
end.
```

Consideraciones...

Un **parámetro por valor** debe ser tratado como una variable local del módulo.

La utilización de este tipo de parámetros puede significar una utilización importante de memoria.

Los **parámetros por referencia** en cambio operan directamente sobre la dirección de la variable original, en el contexto del módulo que llama.

Esto significa que no requiere memoria local.

Recordemos.... ¿Qué tipos de módulos ofrece Pascal?

PROCEDIMIENTOS (PROCEDURE)

FUNCIONES (FUNCTION)

Tienen características comunes, pero ciertas particularidades determinan cual es el mas adecuado para implementar un módulo particular

- ¿El módulo devuelve datos?
  - ¿Cuántos datos devuelve?
    - ¿De qué tipo son los datos que devuelve?
      - ¿Qué tipo de acciones ejecuta el módulo?

¿PROCEDURE?

¿FUNCTION?

¿Cuáles son los aspectos que los diferencian?

- Encabezamiento del módulo
- Invocación
- Lugar donde retorna el flujo de control una vez ejecutado el módulo

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se define el módulo?

```
Function nombre (lista de parametros ): tipo;

Type   ……

Var    ……

begin
    ……..
    nombre := ….;
end;
```

Asignación (obligatoria)

Encabezamiento

Declaración de tipos (opcional)

Declaración de variables (opcional)

Sección de instrucciones

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;

Function cubo (x:integer): integer;
  Begin
    cubo:= x*x*x;
  End;

Var a: integer;

begin
  read (a);
  write (cubo (a));
End.
```

```
Program otro;

Function cubo (x:integer): integer;
  Begin
    cubo:= x*x*x;
  End;

Var a, c: integer;

begin
  read (a);
  c := cubo (a);
End.
```

¿Qué ocurre con el flujo de control del programa?

Luego de ejecutado el módulo, el flujo de control retorna a la misma instrucción de invocación del módulo

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;

 Function cubo (x:integer): integer;
   Begin
     cubo:= x*x*x;
   End;

Var a: integer;

begin
  read (a);
  If (cubo (a) > 100) then …;
End.
```

```
Program otro;

 Function cubo (x:integer): integer;
   Begin
     cubo:= x*x*x;
   End;

Var a: integer;

begin
  read (a);
  while (cubo (a) < 50) do
   ….
End.
```

¿Qué ocurre con el flujo de control del programa?

Luego de ejecutado el módulo, el flujo de control retorna a la misma instrucción de invocación del módulo

¿Qué imprime el programa?

```
Program Ejemplo7;

  Function EsPar (num:integer): boolean;
   var resto:integer;
   begin
    resto := num mod 2;
    if resto = 0 then EsPar := true
                 else EsPar:= False;
   end;

var x: integer;
begin
 read (x);
 if EsPar(x)=true then writeln (´El número ´, x, ´es par´)
                  else writeln (´El número ´, x, ´no es par´)
end.
```