FUNDAMENTOS DE OBJECT PASCAL

1. ASPECTOS GENERALES DE LA SINTAXIS

• Los bloques de código quedan delimitados por `begin` y `end`.

• Se usa punto y coma como separador de sentencias de código.

• Comentarios:
- varias líneas: `(*, *)` o `{, }`
- una línea: `//`

2. DECLARACIONES

• Orden: constantes, tipos y variables (puede alterarse o cambiarse).

• Ejemplo:

```
const
     Pi = 3.14;
     Nombre = 'Pedro';
type
     TVector = array [1..10] of integer;
var
     A: integer;
     V: TVector;
     direccion: string;
     apellidos: string[50];
```

3. IDENTIFICADORES

• Deben ser únicos en su ámbito.

• Formados por letras, números y caracteres subrayado, sin espacios, ni eñes ni acentos. No se distingue entre mayúsculas y minúsculas.

• Los identificadores deben ser descriptivos.

4. TIPOS

4.1. TIPOS PREDEFINIDOS

Nombre	Contenido
Boolean	Un valor lógico: True o False
Char	Un solo carácter
Byte	Un entero entre 0 y 255
SmallInt	Un entero entre –32768 y 32767
Word	Un entero entre 0 y 65535
Integer o LongInt	Un entero entre –2147483648 y 2147483647
Cardinal	Un entero entre 0 y 4294967295
Real o Double	Un real (15/16 dígitos)
Extended	Un real (19/20 dígitos)
String	Una cadena de caracteres

4.2. TIPOS DEFINIDOS POR EL USUARIO

4.2.1. MATRICES

• Definición. Ejemplos:

```
var
   matriz: array [1..25] of integer;
   tabla: array [1..10,1..15] of string;
   tablaAlternativa: array [1..10] of array [1..15]
   of string;
```

• Referencia. Ejemplos:

```
   …
   direccion:= tabla[3,7];
   …
```

4.2.2. ENUMERACIONES

• Definición. Ejemplo:

```
type
   TDiaSemana =
   (lunes, martes, miercoles, jueves, viernes,
   sabado, domingo);
```

• Es un tipo escalar y, por tanto, puede ser el índice de una matriz. Ejemplo:

```
var
 gastos: array[TdiaSemana] of integer;
```

4.2.3. SUBRANGOS

• Su función es limitar los valores asignados a las variables. Ejemplo:

```
type
   TUnoCien = 1..100;
   TDiaLaborable = lunes..viernes;
```

• Obviamente sigue siendo un tipo escalar.

4.2.3. CONJUNTOS

• Para definir variables capaces de contener subconjuntos de un tipo base escalar (de cardinal menor o igual que 256).

• Los elementos del conjunto no están ordenados.

• No se puede acceder a los elementos directamente, sólo se puede comprobar su la pertenencia al conjunto.

• Ejemplo:

```
type
   TDiasSemana = set of TDiaSemana;
var
   partidos, vacio: TDiasSemana;
   dia: TdiaSemana;

   …
   partidos:= [lunes,jueves,domingo];
   vacio:= [];
   if dia in partidos then
   …
```

4.2.4. REGISTROS

• Son tipos compuestos en los que las componentes de sus elementos son de distintos tipos. Ejemplo:

```
type
   TFicha = record
      nombre: string;
      direccion: string;
      edad: integer;
      calificacion: (NP,sus,ap,not,sob,MH)
   end;
```

• Para las referencias a los campos de un registro se emplea el operador de cualificación (operador “.”). Ejemplo:

```
var
   ficha: Tficha;

   …
      ficha.nombre:= ‘Cristina’;
      ficha.direccion:= ‘C. Larios, 1’;
      ficha.edad:= 1;
   …
```

• Las definiciones de tipos se pueden anidar. Ejemplo:

```
var
   agenda: array[1..100] of Tficha;

   …
      agenda[1].nombre:= ‘Cristina’;
   …
```

OBSERVACIÓN:

• En Object Pascal es posible definir **constantes con tipo**, cuyo interés es la definición de constantes de tipos compuestos. Ejemplo:

```
const
   cadenaMes = array [1..12] of string =
      ('Ene','Feb','Mar','Abr','May','Jun','Jul',
      'Ago','Sep','Oct','Nov','Dic');
```

• Este formato de paréntesis y valores separados por comas se emplea también para las constantes de tipo registro.

5. ÁMBITO DE LOS IDENTIFICADORES

• Como en otros lenguajes, se distinguen varios tipos de identificadores:

- Identificadores **globales**: los definidos en la parte de interfaz de una unidad (módulo). Son accesibles por todos los subprogramas de la unidad y por cualquier programa que use la unidad.

- Identificadores **locales**: los definidos en subprogramas. Salvo conflictos, son accesibles en el "interior" del subprograma.

- Identificadores **de unidad**: los declarados en la parte de implementación de una unidad. Son accesibles únicamente por los subprogramas de la unidad.

• Se pueden crear referencias inequívocas para evitar conflictos con identificadores duplicados. Ejemplo:

```
var
   N: integer;

   …
   procedure Lioso;
   var
      N: integer;
   begin
      N:= 5;          //Esta es la variable local
      UnidadU.N:= 3; //y esta es la global
   end;
```

6. EXPRESIONES

6.1. OPERADORES ARITMÉTICOS

• Los habituales: `+, -, *, /, div, mod`

6.2. OPERADORES RELACIONALES

• Los habituales: `=, <>, <, >, <=, >, >=`

6.3. OPERADORES LÓGICOS

• Los habituales: **`not, and, or, xor`**

6.4. OTROS OPERADORES

6.4.1. CONCATENACIÓN DE CADENAS

• Ejemplo:

```
var
   cadena: string;

   …
   cadena:= 'Hola' + 'mundo';
   …
```

6.4.2. OPERADORES DE CONJUNTOS

• Los habituales: + (unión), - (diferencia), * (intersección), **in** (pertenencia). Ejemplo:

```
type
   TdiaSemana =
   (lunes, martes, miercoles, jueves, viernes,
   sabado, domingo);
   TdiasSemana = set of TdiaSemana;
const
   Clase: TdiasSemana = [lunes,miercoles,jueves];
   Tutoria: TdiasSemana = [martes,jueves,viernes];
var
   trabajados, soloClase, ambos: TdiasSemana;
   resultado: boolean;
begin
   trabajados:= Clase + Tutoria;
   soloClase:= Clase - Tutoria;
   ambos:= Clase * Tutoria;
   resultado:= lunes in ambos
 end;
```

6.4.2. OPERADORES DE PUNTEROS

• Los habituales: `@`, `^`, `New` y `Dispose`. Ejemplo:

```
var
   num: integer;
   puntero, puntero2: ^integer;

begin
   puntero:= @num;
   num:= 5  // Es lo mismo que puntero^:= 5
   …
   New(puntero2);
   puntero2^:= 7;
   Dispose(puntero2)
end;
```

6.4.2. ORDEN DE PRIORIDAD

• De izquierda a derecha y con la prioridad habitual, salvo paréntesis. Ordenación de mayor a menor:

1. **not**, @
2. **and**, *, /, **div**, **mod** (multiplicativos)
3. **or**, **xor**, +, - (aditivos)
4. =, <, <=, >, >=, <>, **in** (relacionales)

7. ESTRUCTURAS DE CONTROL

7.1. CONDICIONALES

• Con el significado conocido:

```
if condición then
   bloque de instrucciones
else
   bloque de instrucciones;
```

• Selección múltiple:

```
case expresión escalar of:
   valor1: bloque de instrucciones 1;
   …
   valorN: bloque de instrucciones N
else
   bloque de instrucciones alternativo;
end;
```

7.2. ITERATIVAS

• Con el significado conocido:

```
for contador:= valorInicial to valorFinal do
   bloque de instrucciones;

for contador:= valorInicial downto valorFinal do
   bloque de instrucciones;

while condición do
   bloque de instrucciones;
```

```
repeat
   bloque de instrucciones
until condición;
```

• Instrucciones de control de bucles:

- `continue`: no se ejecutan el resto de las sentencias del bucle, transfiriendo el control a la primera instrucción (previa comprobación de la condición)
- `break`: finaliza la ejecución del bucle.

• Ejemplo:

```
var
   contador, dato, suma: integer;
   puntero, puntero: ^integer;

begin
//sumaremos numeros menores que 9
   suma:= 0;
   for contador:= 1 to 100 do
   begin
      dato:= PideNumero;
      //si es cero salimos del bucle
      if dato = 0 then
          break;
      //si es mayor que 9 lo ignoramos
      if dato > 9 then
          continue;
      //en caso contrario se acumula
      suma:= suma + dato
   end
 end;
```

8. SUBPROGRAMAS

• Procedimientos:

```
procedure NombreProcedimiento (listaParametros);
Declaraciones;
begin
   bloque de instrucciones
end;
```

• Funciones:

```
function NombreFuncion (listaParametros): tipoDevuelto;
Declaraciones;
begin
   bloque de instrucciones
end;
```

• Se pueden pasar parámetros por valor y por referencia. Ejemplo:

```
procedure Cuadrado (factor: integer;
                    var resultado: integer);
begin
   resultado:= factor * factor;
end;
```

• Para los parámetros de salida de las funciones se emplea la variable especial (local) `result`. Ejemplo:

```
function Cuadrado (factor: integer): integer;
begin
   result:= factor * factor;
end;
```

9. DIFERENCIAS ENTRE MODULA-2 Y OBJECT PASCAL

MÓDULOS

• El módulo de programa equivale al programa principal (proyecto):

MODULA-2

```
MODULE Test;
VAR
  …
BEGIN
   …
END Test.
```

Object Pascal

```
Program Test;
var
…
begin
     …
end.
```

• Módulo = unidad.

• Partes de un módulo:

Módulo de definición	Interfaz de la unidad
Módulo de implementación	Implementación de la unidad

· **IMPORT = uses**

TIPOS DE DATOS

• El tipo **cardinal** de MODULA-2 equivale al tipo **word** de Object Pascal.

IDENTIFICADORES

• Object Pascal no es sensible a las mayúsculas.

• En Object Pascal se puede emplear el carácter subrayado en los identificadores.

COMENTARIOS

• En Object Pascal se emplean `(*, *)` o `{, }` y `//,` mientras que en MODULA-2 las llaves se emplean para los conjuntos.

PROCEDIMIENTOS ESTÁNDAR

• En Object Pascal se emplean `Write y WriteLn` para caracteres, números y cadenas, mientras que en MODULA-2 se tienen `Write, WriteLn, WriteString, WriteInt, WriteCard` y `WriteReal.`

INSTRUCCIONES ITERATIVAS

• Diferente uso de **`begin-end`** en la instrucción **`while`**:

MODULA-2

```
WHILE x < 100 DO
…
…
END;
```

Object Pascal

```
while x < 100 do
begin
    …
end;
```

• La sintaxis de la instrucción **for** se ha extendido en MODULA-2:

```
FOR x:= 0 TO 10 BY 2 DO
   WriteCard(x,2);
END;
```

• En MODULA-2 no existe la posibilidad de usar **downto** en la instrucción **for**:

```
FOR x:= 10 TO 0 BY -1 DO
   WriteCard(x,2);
END;
```

• En Object Pascal no existe la construcción **LOOP-EXIT**:

```
LOOP
   Read(ch);
   …
   IF ch = 'q' THEN
     EXIT
   END;
END;
```

INSTRUCCIONES DE SELECCIÓN

• En MODULA-2 la instrucción **IF** siempre termina con **END**:

MODULA-2

```
IF ch = '+' THEN
     Sumar(a,b)
ELSE
     Restar(a,b)
END;
```

Object Pascal

```
if ch = '+' then
     Sumar(a,b)
else
     Restar(a,b);
```

- Instrucción **case** con varias sentencias en el mismo caso:

MODULA-2

```
CASE eleccion OF
     '1': CorrecOrtog;|
     '2': Reset;
          Salvar;|
     '3': …
ELSE
     Error;
END;
```

Object Pascal

```
CASE eleccion OF
     '1': CorrecOrtog;
     '2': begin
               Reset;
               Salvar;
          end;
     '3': …
ELSE
     Error;
END;
```

• En Object Pascal no se puede emplear **ELSIF** en las instrucciones **IF**:

MODULA-2

```
IF condicion A THEN
     Bloque de instrucciones A
ELSIF condicion B THEN
     Bloque de instrucciones B
ELSE
     Bloque de instrucciones por defecto
END;
```

Object Pascal

```
if condicion A then
     Bloque de instrucciones A
else if condicion B then
     Bloque de instrucciones B
else
     Bloque de instrucciones por defecto
end;
```

FUNCIONES

• En Object Pascal se utiliza la palabra reservada **`function`** para subprogramas que devuelven valores:

MODULA-2

```
PROCEDURE Muestra (cont: INTEGER): BOOLEAN;
BEGIN
     IF cont < 100 THEN
          RETURN TRUE
     ELSE
          RETURN FALSE
     END;
END Muestra;
```

Object Pascal

```
function Muestra (cont: integer): boolean;
begin
     if cont < 100 then
          result:= true
     else
          result:= false
end;
```