

CURSO DE

PROGRAMACIÓN

EN

PASCAL

ÍNDICE

1. Introducción
2. Constantes y variables
3. Definiciones y declaraciones
4. Tipos de datos
 - 4.1. Integer
 - 4.2. Real
 - 4.3. Boolean
 - 4.4. Char
 - 4.5. String
 - 4.6. Array
5. Declaración de tipos de datos
 - 5.1. Records
 - 5.2. Conjuntos
 - 5.3. Subrango
6. Sentencias de control de flujo
 - 6.1. Estructuras condicionales
 - 6.2. Bucles
 - 6.2.1. Bucle “**for**”
 - 6.2.2. Bucle “**while**”
 - 6.2.3. Bucle “**repeat**”
7. Construcción de programas en Pascal
 - 7.1. Diagramas básicos
 - 7.2. Diagramas de control de flujo
 - 7.2.1. Secuencial
 - 7.2.2. If – then – else
 - 7.2.3. Bucle
 - 7.3. Entrada / salida
 - 7.3.1. Write()
 - 7.3.2. Read()
8. Procedimientos y funciones
9. Archivos
 - 9.1. Procedimientos estándar para tratamiento de archivos
10. Punteros
11. Apéndice A – Palabras reservadas en Pascal
12. Apéndice B – Funciones y procedimientos predefinidos

INTRODUCCIÓN

El lenguaje Pascal tuvo sus orígenes en el lenguaje de programación ALGOL hacia 1965. La primera versión de este lenguaje salió a la luz en el 68. Después de varios años de realce salió en 1970 el primer compilador operativo. Años más tarde y después de varias revisiones se desarrolla la primera estandarización en el año 1980 promovida por la BCS.

A partir de este lenguaje de programación se desarrollaron otros lenguajes cada vez más adecuados al usuario ya que los anteriores lenguajes eran demasiado especializados y complejos.

Dentro del lenguaje Pascal hay ciertos puntos característicos que es necesario saber. Entre ellos destacan:

- La declaración de variables es obligatoria.
- Hay palabras reservadas que no pueden emplearse para la declaración de variables u otros identificadores.
- Las sentencias se separan únicamente por punto y coma, por lo tanto es un lenguaje de formato libre.
- Hay varios tipos de datos estándar como el entero, real, etc. Todos ellos se pueden combinar entre sí.
- El manejo de la memoria es dinámico (no es necesaria la reserva de la misma por parte del usuario).
- Los arrays (tipo de datos) son estáticos y se declaran en el tiempo de compilación.
- Los parámetros de las funciones pueden ser del tipo referencia o valor.

Todos estos conceptos se explicarán en capítulos posteriores.

CONSTANTES Y VARIABLES

Un dato refleja normalmente una medida del mundo físico que generalmente debemos introducir en nuestro programa que a su vez está almacenado en el ordenador. Estos datos se pueden pasar al programa de varias formas entre ellas mediante el teclado, el ratón, disco duro, etc.

El mayor problema que existe es confundir un dato con otro, por eso el Pascal clasifica cada dato dentro de una familia de un tipo determinado. Para hacer esta clasificación hay unas reglas rígidas que es necesario seguir para conseguir un programa en Pascal.

Definamos antes de nada unos conceptos básicos:

- ✓ **Identificador:** el Pascal, al ser un lenguaje simbólico con uso parecido al álgebra, tiene una forma de identificar las variables dándoles el nombre que el programador considere oportuno (dentro de unas restricciones). Este nombre de variable o constante es el “identificador”. El identificador debe seguir una secuencia de caracteres comenzando por una letra y seguida de 0 o más letras o dígitos decimales.
- ✓ **Número entero:** Se define un número entero como una secuencia de dígitos decimales. Un dígito decimal es el carácter 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Un número entero debe constar de por lo menos un dígito.
- ✓ **Constantes y variables:** Los datos de un programa se pueden clasificar con constantes y variables. Las constantes, como su nombre indica, son datos que no varían durante la ejecución del programa. Cada programa tendrá su propio tipo de constantes. Las constantes pueden tener su nombre o identificador para poder utilizarlas a lo largo del programa. Las variables son tipos de datos que cambian a lo largo de la ejecución del programa. Estos tipos de datos se les llama variables. Como ejemplo poner un contador que con cada cierta acción se incremente o se decremente. Las variables pueden ser de ciertos tipos (que veremos más adelante) para almacenar los valores correspondientes a los tipos de dichas variables. Las variables se consideran como depósitos para almacenar valores.
- ✓ **Operador:** un operador es un símbolo que se usa conjuntamente con identificadores (variables o constantes) o con números que después de ser evaluado durante la ejecución del programa produce un valor de uno u otro tipo. Un ejemplo de operador es el signo más (+) que junto con dos números produce la suma de estos. Este identificador también sirve para sumar variables de tipo entero o real (veremos más adelante estos tipos), con lo que produce un valor entero o real.

DEFINICIONES Y DECLARACIONES

En Pascal cada identificador debe declararse al principio del programa. Puede haber varias declaraciones de variables o constantes separadas por un punto y coma. Un ejemplo de declaración de constantes sería el siguiente:

```
const VelocMax = 150.25;
      DiasSem = 7;
      MaxCount = 1024;
```

Donde **const** es una palabra reservada de Pascal que se usa para la declaración de constantes y sólo se puede usar para eso. Para la declaración de constantes sólo es necesario poner una vez la palabra reservada **const**. Después se escribe el identificador de la constante (el nombre) seguido de un signo igual y el valor que se le quiere asignar.

Finalmente se pone punto y coma para terminar la declaración de dicha constante y permitir la declaración de más de las mismas. El identificador *VelocMax* es de tipo real mientras que los otros dos son de tipo entero. Esto quiere decir que el tipo de las constantes se identifica por el valor que toman.

La declaración de variables se hace de forma parecida. Este es un ejemplo de cómo se declararían variables en Pascal:

```
var Contador: Real;  
    dia1, dia2: Integer;  
    Verdadero: Boolean;
```

La palabra reservada **var** se usa para la declaración de variables y para nada más. Con una vez que se escriba la palabra **var** es necesario para la declaración de todas ellas. Después de la palabra **var** viene el identificador de la variable, que es el nombre que le damos para identificarla en el resto del programa, seguido de dos puntos y el tipo de datos que va a almacenar (que veremos en posteriores capítulos). Finalmente, y como en toda sentencia de Pascal, el punto y coma para identificar el final de dicha sentencia.

Después de la declaración de variables es necesario saber la sentencia más sencilla y quizá más importante dentro del lenguaje Pascal. Esta es la sentencia de asignación cuya sintaxis es la siguiente:

variable := expresión;

Esta sentencia produce dos acciones:

- Produce un valor de la expresión que está a la derecha del signo := (esta expresión puede ser otra variable, una constante, un valor explícito o una expresión que sea necesaria su evaluación).
- Asigna el valor de la expresión a la variable que está en lado izquierdo del signo “:=”.

Algunos ejemplos pueden ser:

```
Dia := 25;  
Mes := MesSiguiente - MesAnterior;  
Area := 2 * pi * r ^ 2;
```

Conviene distinguir entre los signos “=” (igual) y “:=” (asignación). Con igual definimos constantes (y más tarde haremos comparaciones) y con el signo “:=” estamos haciendo asignaciones a variables.

En la asignación sólo se acepta si los tipos de datos y el tipo de la variable son el mismo, sino el compilador dará un error.

TIPOS DE DATOS

Dentro de la declaración de variables y constantes hemos visto que las constantes saben de qué tipo son por el dato que contienen y que las variables son declaradas de un tipo de datos determinado para poder almacenar valores del tipo de dato declarado. Ahora veremos los diferentes tipos de datos con los que se pueden declarar las variables y pueden ser las constantes.

Datos de tipo Integer:

Este tipo de datos nos permite almacenar valores de tipo entero. Valores desde menos infinito ($-\infty$) hasta más infinito ($+\infty$) incluyendo el cero. Como la representación de infinitos números dentro del ordenador es imposible, un entero se representa con 32 bits, por tanto el rango va desde $-2.147.483.648$ hasta $+2.147.483.647$.

Con este tipo de datos se pueden hacer las operaciones normales de los enteros, como la suma, la resta, la multiplicación, la división entera, etc

Para realizar una operación se hace mediante las expresiones, que son un conjunto de caracteres que, siguiendo las reglas del Pascal, producen un valor que puede ser asignado a una variable (nunca a una constante) o usado por otra expresión para producir otro valor diferente. Las operaciones siguen un orden de prioridad que es el siguiente:

- Se efectúan las operaciones de multiplicación y división empezando por la izquierda.
- Después las de suma y resta empezando por la izquierda.
- Si hay paréntesis se evalúan primero de acuerdo con las reglas anteriores.

Los valores de una expresión se almacenan generalmente en una variable para su posterior uso. Si la variable que queremos usar es entera, la expresión que queremos construir debe ser también entera (debe producir un valor entero).

Algunos ejemplos de expresiones pueden ser los siguientes:

```
x + y
z - NumMeses
k * v * i
```

Donde todos los nombres son identificadores de variables. Para hacer la asignación se construye una expresión a partir de las anteriores y con la sentencia de asignación. La declaración de las variables que se van a usar se hace de la siguiente forma así como la expresión:

```
var x, y: Integer;
x := x + y;
```

Donde en la variable de nombre “x” se asigna el valor producido al evaluar la expresión “x + y” donde ambos identificadores son enteros para producir un valor entero. Los valores de las variables x e y han sido previamente asignados.

*Datos de tipo **Real**:*

Los datos de tipo real son números al igual que los de tipo integer pero con decimales. Además tienen las mismas operaciones excepto la división entera que pasa a ser la división real. Los números reales también se representan con 32 bits (también pueden representarse con 64 bits para hacerlos de precisión doble) y su rango va desde menos infinito hasta más infinito teniendo en cuenta que entre el 0 y el 1 hay infinitos valores. Dentro del ordenador los rangos son desde 3.40282347E+38F hasta 1.17549435E-38F.

La declaración tanto de constantes como de variables se hace de la misma forma que con los enteros:

```
const float = 3.25482;  
      pi = 3.1415926535897932384626433832795;  
  
var  veloc, accel: Real;
```

*Datos de tipo **Boolean**:*

Los datos de tipo Boolean son los que se usan para producir operaciones booleanas, que son operaciones que usan sólo dos valores: **True** y **False**. Estos valores son útiles en ciertas operaciones como para tablas de verdad con operadores **or**, **and** y **not**. También los valores True y False son producidos al hacer evaluaciones de expresiones como comparaciones: Por ejemplo:

```
var b: Boolean;  
  
b := x < y;  
b := True or False;  
b := (x >= y) and False;
```

En la variable “b” de tipo booleano habrá uno u otro valor dependiendo de si al contenido de la variable x es menor que el de la variable y, en el segundo caso dependiendo de la tabla de verdad del **or** y en el tercer caso dependiendo de que valor tiene la expresión entre paréntesis y luego mediante un **and** con False. Si esto es así en “b” estará el valor True y sino estará el valor False. Las palabras or, and y not son palabras reservadas en Pascal. Estas palabras son operadores específicos para valores booleanos. Para ver su efecto cabe fijarse en la siguiente tabla conteniendo las tablas de verdad de estos operadores:

p	p	p or q	p and q	not p
True	True	True	True	False
True	False	True	False	False
False	True	True	False	True
False	False	False	False	True

Hay otros operadores para enteros que producen resultados booleanos. Estos son:

- < (menor que, en Pascal <)
- > (mayor que, en Pascal >)
- = (igual, en Pascal =)
- ≤ (menor o igual que, en Pascal <=)
- ≥ (mayor o igual que, en Pascal >=)
- ≠ (distinto, en Pascal <>)

Todas las aplicaciones que se le pueden dar a las variables booleanas se basan en el álgebra de Boole con las correspondientes reglas que aquí no vamos a explicar.

Tipo de dato Char (carácter):

Debido a la gran comunicación que debe tener el usuario con el ordenador es necesario tener algún método para que dicha comunicación se parezca al lenguaje natural humano. Para ello, el Pascal implementa el tipo de datos carácter (char) que es un tipo que almacena un carácter. El ordenador lo almacena como un dato numérico aunque su representación en pantalla sea como un carácter de alfabeto humano.

Los caracteres reconocidos por el ordenador pueden variar, ya que hay varias codificación que tienen más o menos caracteres (según el idioma, ya que en español están la “ñ” mientras que en el inglés no). Dos de ellas (y las primeras que se desarrollaron) son el ASCII que consta de 128 caracteres (7 bits por carácter) y el EBCDIC de 256 caracteres (8 bits por carácter). Actualmente se están intentando la unificación de todas las tablas de caracteres con el Unicode (también conocido como UTF o UTF-8) que consta de 16 bits por carácter, con lo que se pueden almacenar casi todos los caracteres de los idiomas más importantes. Por desgracia, el Pascal todavía no reconoce UTF-8.

Un carácter en Pascal se representa entre comillas simples: ‘a’. Un ejemplo de la definición de constantes y variables de tipo carácter (tipo char) puede ser la siguiente:

```

const blanco = ' ';
      número = '7';
      letra = 'x';
      ampersand = '&';

var let, simb, car: char;

let := 'r';
car := 'm';
simb := '%';

```

Las constantes son blanco que es el carácter blanco (espacio), número, que es el carácter 7 (NO es el número 7 de tipo entero), letra que es la “x” y el ampersand que es el símbolo “&”. Después se declaran tres variables (let, simb y car) de tipo carácter que se van a encargar de almacenar caracteres (sólo uno, para almacenar más de uno se verá otra estructura posteriormente). A estas variables se les ha asignado un carácter. En el caso de que al asignación fuese un entero u otro tipo de dato el compilador daría un error.

*Tipo de dato **String** (cadena de caracteres):*

Cómo el manejo de un solo carácter de cada vez es un poco complicado, el Pascal implementa lo que se llama string, que es una tira de caracteres delimitada por las mismas comillas simples por las que se delimitaba el carácter, por tanto para declarar constantes y variables de tipo string se hace de la siguiente forma:

```
const Nombre = 'Manolo';
      Apellido = 'Lobato';

var Trab, Jefe: String;

Trab := 'Luis Frade';
Jefe := Trab + ' Carballo';
```

Las operaciones con strings son simples. Hay una comparación en la que se devuelve un valor booleano, una asignación típica y una suma con el operador “+”. En la variable Jefe a la que se le ha asignado la variable Trab que vale “Luis Frade” más la cadena de caracteres “ Carballo” quedaría como “Luis Frade Carballo”.

Para el tratamiento de strings hay operadores genéricos, como el igual, los signos de mayor y menor que, distinto, etc., y también hay ciertas funciones estándar provistas por el Pascal para el manejo de cualquier string, que pueden ser desde comparaciones optimizadas (en lugar de usar el signo =) hasta la supresión de un número determinado de caracteres dentro de una string dando el carácter principal y el número de ellos que hay que suprimir. Estas funciones las veremos más adelante. Los operadores de comparación ordenan las cadenas de caracteres según el orden lexicográfico, o sea, el orden del diccionario, teniendo en cuenta que la “ñ” no está en la posición que nosotros estamos acostumbrados.

*Tipo de dato **Array** (matriz):*

Para darle mayor potencia al Pascal se ha implementado un tipo de datos que no es tal. Quiero decir, el tipo de datos array puede estar compuesto por cualquiera de los tipos de datos explicados anteriormente.

Un array es una matriz de elementos, todos iguales (del mismo tipo pero que pueden contener valores diferentes) y de tamaño prefijado en tiempo de compilación. Un array es una representación dentro del ordenador de las matrices del álgebra. Los arrays pueden ser unidimensionales, bidimensionales, ..., n-dimensionales. Un array se declara de la siguiente forma:

```
var NombreArray: array[1..25] of Integer;
```

En esta declaración “NombreArray” es el identificador el array. La palabra “array” es una palabra reservada en Pascal por lo que solo se puede usar para declarar arrays. El tamaño del array es de 25 elementos, el array es unidimensional y el tipo de los datos que va a almacenar son enteros. Para acceder a los elementos del array se usa de la siguiente forma:

```
var x: Integer;  
NombreArray[20] := 12;  
x := NombreArray[16];
```

Al elemento 20 del array se le asigna el valor 12. Luego a la variable x se le asigna el valor de la posición del array 16 que puede contener cualquier valor de tipo entero. Las posiciones del array funcionan como variables normales con la diferencia de que todas las variables están condensadas bajo un mismo nombre y a las que se puede acceder mediante su posición dentro del array. La primera posición dentro del array es la 0, y la última dentro de este array es la 24 (si pones la posición 25 da error). En ciertos compiladores la primera posición es la 1 y la última es la 25, en el array de ejemplo.

La declaración de un array multidimensional se hace de la siguiente forma:

```
var multi: array[1..25,1..30,1..15] of char;
```

Aquí tenemos un array tridimensional. La primera fila tiene 25 elementos, la segunda 30 y la tercera 15. Para acceder a cualquiera de sus elementos se hace de la siguiente forma:

```
var x: char;  
x := Multi[12,26,3];  
Multi[5,7,14] := 'm';
```

Los arrays pueden ser declarados de cualquiera de los tipos anteriormente explicados, pero todas sus dimensiones han de ser del mismo tipo.

DECLARACIÓN DE TIPOS DE DATOS

Aparte de los tipos de datos anteriormente citados, el programador puede declarar sus propios tipos de datos. Para ello se usa la palabra reservada en Pascal “**type**”. Dentro de esta declaración existe otra palabra reservada que es “**record**” que se usa para declarar estructuras de tipos de datos.

La palabra “**type**” se usa de la siguiente forma:

```
type Cadena = array[1..256] of char;
```

Con esta declaración hemos declarado un tipo de datos nuevo que se llama cadena que es un array de 256 elementos de tipo carácter. Para hacer uso de el tipo creado se declaran variables de dicho tipo:

```
var Cadena1: Cadena;
```

Tenemos una variable Cadena1 de tipo cadena, o sea, Cadena1 es un array de 256 elementos de tipo carácter. Esto es útil para no tener, cada vez que se desea declarar un variable como Cadena1, que escribir toda la parafernalia del array.

Existe también otra palabra reservada que es “**record**” para crear tipos de datos con varios tipos de datos diferentes dentro de la estructura. Este tipo de datos ser llama registro. Para crear un tipo registro se usa la siguiente sintaxis:

```
type Fecha = record
    Dia: Integer;
    Mes: Integer;
    Anno: Integer;
end;
```

Después de hacer la declaración del nuevo tipo se declara una variable y se accede a los campos de dicho registro de la siguiente forma:

```
var Fecha1: Fecha;

Fecha1.Dia := 28;
Fecha1.Mes := 9;
Fecha1.Anno := 2000;
```

Para acceder a un campo de un registro (cualquiera) se pone el nombre del registro (de la variable de tipo del registro) seguido de un punto y el nombre del campo al que se quiere acceder.

También puede haber arrays de records y records con arrays en alguno de sus campos. La declaración y el acceso a los mismo se realiza de la siguiente forma:

```
type Ejemplo = record
    Num: Integer;
    Text: String;
    Arr: array[1..16] of char;
end;

ArrayRecords = array[1..5] of Ejemplo;

var Ejemplo1: Ejemplo;
    Array1: ArrayRecords;

Ejemplo1.Num := 28;
Ejemplo1.Text := 'Hola mundo';
Ejemplo1.Arr[10] := 'z';

Array1[0].Num := 9;
Array1[4].Text := 'Esto es un record en un array';
```

```
Array1[2].Arr[8] := 'ñ';
```

En los ejemplos anteriores se accede tanto al array dentro de los registros encuentran en otro array, como a los registros dentro del array. El array dentro del registro es otro campo del mismo, por tanto la forma de acceder a él es la misma que para cualquier campo y cualquier array.

Otro tipo de datos definido por el usuario es el tipo de datos enumerado (o escalar). Este tipo de datos se declara también con la palabra reservada “type” y lo que se hace es crear un tipo de datos con campos que no tienen relación con los tipos de datos anteriormente vistos. Por ejemplo:

```
type figura = (cuadrado, rombo, circulo);
moneda = (euro, dollar, peseta, lira, marco, libra, rublo);
color = (rojo, verde, azul);

var f: figura;
m: moneda;
c: color;

f := cuadrado;
m := peseta;
c := azul;
```

La definición de estos tipos de datos escalares es para crear un conjunto de elementos para facilitar su tratamiento. El compilador asigna valores consecutivos, comenzando en 0, a los elementos del tipo nuevo creado. Como el tratamiento con números es más difícil para los usuarios, es por eso por lo que el Pascal provee de este método para tratar conjuntos de datos.

Con estos tipos de datos se puede hacer un conjunto (una estructura que pueda contener 0, 1 o varios de estos tipos de datos) para hacer más fáciles las operaciones. Este conjunto se crea con la palabras reservadas “set” y “of”. Por ejemplo:

```
var conj: set of moneda;

conj := libra;
```

El conjunto “conj” contiene el elemento “libra”;

```
conj := conj + [peseta];
```

“conj” contiene ahora libra y peseta;

```
conj := conj - [libra];
```

“conj” ahora tiene sólo peseta;

```
var b: boolean;

b := [peseta] in conj;
b := [peseta] in [libra,marco,dollar,euro];
```

Con la palabra reservada “in” se comprueba si un elemento está dentro de un conjunto. En la primera expresión anterior, el booleano “b” será True ya que peseta se encuentra en el conjunto “conj”, en cambio en el segunda expresión el valor de “b” será

False ya que el elemento “peseta” no se encuentra en el conjunto dado. El tema de conjuntos tiene muchas más operaciones y es más complicado de lo que aquí se muestra, pero esto no será explicado ya que este no es el cometido de este curso.

Finalmente, el último tipo de datos que veremos es el tipo de datos subrango o intervalo. Este tipo de datos sirve para limitar a una variable en un número de valores posibles. La declaración es la siguiente:

```
var v: 1..100;
```

Donde la variable “v” sólo puede tomar valores entre 1 y 100. Esto también se puede declarar como un tipo y luego declarar una variable de ese tipo de la forma:

```
const Max = 100;  
      Min = 1;  
  
type rango = Min..Max;  
      digit = '0'..'9';  
  
var v: rango;  
    d: digit;
```

SENTENCIAS DE CONTROL DE FLUJO

Después de haber analizado y explicado de una forma más o menos comprensible los tipos de datos que el Pascal utiliza para manejar la información de entrada y de salida, ahora es necesario aprender ciertas estructuras básicas que provee el Pascal para hacer más sencilla la creación de programas más o menos complejos.

La estructura más sencilla que hemos visto hasta el momento es la estructura de asignación que se encargaba de “meter” valores dentro de las variables de algún tipo predefinido. Esta asignación se realiza mediante el signo “:=”.

A partir de este momento se van a ver ciertas estructuras de control de flujo que lo que hacen es controlar la ejecución de un programa en Pascal. Estas estructuras se pueden resumir en condicionales y bucles.

Estructuras condicionales: if

La palabra reservada de Pascal “**if**” se utiliza para evaluar expresiones y dar como resultado un valor booleano (True o False) y en consecuencia, si dicho valor es verdadero, se ejecutan una serie de sentencias y si el valor es falso se ejecutan otras. Gracias a esto, la ejecución de un programa se bifurca dependiendo de ciertos parámetros, como puede ser la intervención del usuario, el resultado de alguna expresión, etc.

La sentencia “**if**” se construye de la siguiente forma:

if <condición> **then** <resultado si condición verdadera>;

if <condición> **then** <resultado si condición verdadera> **else** <resultado si falsa>;

En el primer caso, si la condición es verdadera, se ejecutan las sentencias predefinidas después del “**then**”, si es falso se sigue con la ejecución normal del programa. En el segundo caso, si la condición es verdadera, se ejecutan las sentencias después del “**then**”, mientras que si es falso, se ejecutan las sentencias que están después del “**else**”. Una vez ejecuta esto se continúa con la ejecución normal del programa. Como ejemplo de sentencia “**if**” se podría poner el siguiente:

```
var a, b: Boolean;

a := True;

if a = True then b := False else b := True;
```

En esta sentencia, si la variable de tipo booleano “a” es verdadera, a la variable “b” se le asigna el valor False mientras que si “a” es False, a “b” se le asigna True.

En el caso de que dentro del bloque de sentencias a ejecutar como resultado de la comparación sea mayor de una (mayor de una asignación y otras sentencias de control como pueden ser otro u otros “if”), dicho bloque de sentencias se pone entre las palabras reservadas de Pascal “begin” (para comenzar) y “end” (para terminar):

```
var x: Integer;

if a = True then begin
  B := False;
  X := 1;
end else begin
  B := True;
  X := -1;
end;
```

Dentro de un programa, la sentencia condicional es una de las más usadas ya que con ella es con la que más se interactúa con el usuario, ya que dependiendo de la entrada de dicho usuario y haciendo las correspondientes comparación mediante esta sentencia, se consiguen los resultados que el usuario final desea (a modo de opciones para conseguir un resultado determinado).

Bucles:

Después de conseguir comparación para bifurcar el flujo de un programa y llevarlo según las exigencias del usuario, ahora es necesario aprender los bucles.

Los bucles son estructuras de control provistas por Pascal para ahorrar tiempo a la hora de ejecutar una sentencia varias veces. Un bucle da un mecanismo genérico para la ejecución repetitiva de sentencias.

Un bucle se caracteriza por dos puntos:

- La sentencia que se repite.
- La prueba de condición para hacer terminar el bucle.

Una forma genérica de definición de un bucle puede ser la siguiente:

- Prueba de condición. Si False ir a paso d).*
- Ejecutar la sentencia o sentencias.*
- Ir al paso a)*
- Continuar con el resto del programa.*

Hay varios tipos de bucles que todos hacen lo mismo pero su diferencia radica en la forma de comparación. A continuación definiremos los tres tipos que implementa el Pascal:

Bucle “for”:

El bucle “for” es una sentencia repetitiva cuya principal característica es que el número de veces que se repite es constante (esto no es cierto del todo). Para ello, la declaración de dicha estructura es de la siguiente forma:

for <variable de control> := <valor inicial> **to** <valor final> **do** <sentencias>;

Este es un for ascendente. La forma descendente es la siguiente:

for <variable de control> := <valor inicial> **downto** <valor final> **do** <sentencias>;

La variable de control es la variable que se va a incrementar automáticamente durante la ejecución del bucle para saber la condición en la que debe terminar el bucle. El valor inicial es el valor con el que se inicia la variable de control y el valor final es con el que se compara para saber la condición de terminación. Las sentencias son las ejecutadas en cada pasada del bucle. Por tanto, un bucle “for” es una estructura de Pascal que hace que se repita una sentencia o número de sentencias un número determinado de veces. Esto, como antes se ha dicho, no es cierto del todo, ya que si en las sentencias que ejecuta el bucle se pone una que modifique la variable de control, el número de veces que se ejecutará dicho bucle no será constante.

Si en el bucle “for” se pone la palabra reservada “to”, la variable de control se incrementará, mientras que si se pone “downto”, la variable de control se decrementará. Esto se ha hecho con vistas a la utilización de la variable de control dentro de las sentencias que ejecuta dicho bucle.

Un ejemplo de bucles “for” puede ser:

```
var a: array[1..100] of Integer;
    b: array[1..100] of String;
    i: Integer;

for i := 1 to 100 do a[i] := 0;

for i := 0 to 100 do begin
  a[i] := 0;
```

```
b[i] := '';  
end;
```

El primer bucle se ejecuta 100 veces inicializando cada elemento del array “a” al valor 0. El segundo realiza el mismo número de pases que el anterior pero ejecuta dos sentencias por cada pase. Inicializa los valores del array “a” al valor 0 y los valores del array “b” a la cadena vacía (‘). En el segundo caso, como el número de sentencias es mayor de uno, se han usado las palabras reservadas de Pascal “**begin**” y “**end**” para agrupar sentencias.

Bucle “while”:

La sentencia “while” del Pascal es una de sus sentencias más básicas. Esta estructura también hace un bucle, como la sentencia “for”, con la única diferencia que ésta ejecuta la sentencia hasta que la condición de evaluación no se cumpla. Aquí no incremento de variable sino que se pone una condición de la misma forma que en una sentencia “if”.

La estructura de la sentencia “while” es la siguiente:

```
while <condición> do <sentencia>;
```

La condición dentro de esta sentencia es de la misma forma que con la sentencia de comparación genérica “if”, por tanto, un ejemplo puede ser:

```
var n: Integer;  
  
n := 0;  
  
while n < 100 do n := n + 5;
```

El bucle “**while**” se ejecuta mientras la variable “n” sea menor que 100. En cada pase del bucle, la variable “n” se incrementa en 5 unidades. En el caso de querer ejecutar más de una sentencia, se usarían las palabras reservadas de Pascal “**begin**” y “**end**”.

Bucle “repeat”:

El bucle “repeat” es casi igual al bucle “while”. Las únicas diferencias radican en la declaración y en la forma de actuar en la primera pasada. La declaración es la siguiente:

```
repeat <sentencias> until <condición>;
```

Las sentencias son las que se ejecutan y la condición es la condición de parada del bucle. La diferencia principal respecto al bucle “while” es que el bucle “while” lo primero que hace es la comparación de la condición, por tanto puede que el bucle “while” no se ejecute ninguna vez si la condición inicial es falsa, mientras que el bucle “repeat” lo primero que hace es ejecutar la sentencia o sentencias, por tanto, el bucle

“repeat”, siempre se ejecuta una vez al menos. Un ejemplo de bucle “repeat” sería el siguiente:

```
var x: integer;  
  
x := 0;  
  
repeat x := x + 2 until x > 100;
```

Este ejemplo hace lo mismo que el ejemplo anterior del bucle “while”, pero lo que hay que observar es que el formato de la condición es el contrario al bucle “while”. Si en “while” se ponía $n < 100$, aquí se pone $n > 100$. Como hasta ahora, si el número de sentencias es mayor que una, se usarán las palabras reservadas “begin” y “end” para la agrupación de sentencias.

Una vez concluida la especificación de tipos y de sentencias de control de flujo de programa, pasaremos al desarrollo de programas en Pascal propiamente dichos.

CONSTRUCCIÓN DE PROGRAMAS EN PASCAL

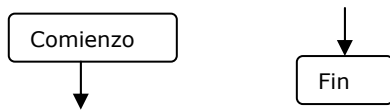
Antes de empezar a construir un programa en cualquier lenguaje de programación, es necesario una planificación previa del mismo mediante un diagramas llamados diagramas de flujo de datos, diagramas de estructuras, diagramas de control, diccionario de datos, etc. Como el desarrollo de programas pequeños es sencillo, no se hace necesario usar estas herramientas de forma muy exhaustiva, pero siempre hay que tener en cuenta que estos diagramas son los que definen la estructura y el funcionamiento del programa conforme a las especificación que el usuario o persona que quiere el programa nos ha dado.

Los diagramas más usados, y que vamos a ver un poco por encima, son los diagramas de “Chapin”, aunque en realidad fueron desarrollados por Nassi y Schneiderman. Estos diagramas constan de varios bloques básicos de la programación. Estos no son demasiados, ya que según la teoría de Bohm y Jacopini, todo programa se puede construir con tres estructuras básicas, que son:

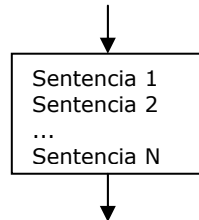
- ✓ Una sentencia de acción secuencial que no modifica el flujo de ejecución del programa (sentencia de asignación, entrada, salida, ...).
- ✓ Una sentencia condicional **if- then -else** que según sea el valor de la comparación ejecutará una u otra de las alternativas.
- ✓ Una sentencia de bucle que permite ejecutar una sentencia o grupo de sentencias un número determinado de veces (**while, repeat, for**).

Estas estructuras son representable mediante diagramas de Chapin. Antes de definir las, diremos cuales son los principales bloques de control:

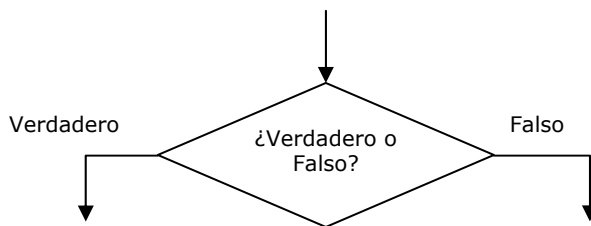
- *Comienzo y final de programa:*



- *Bloque de proceso o acción (rectángulo con descripción de la acción):*



- *Bloque de decisión (rombo con dos salidas según el resultado de la comparación):*



Aunque las estructuras de control de estos diagramas son varias, se enumerarán a continuación aunque no se mostrará el gráfico correspondiente ya que este no es el cometido de este curso:

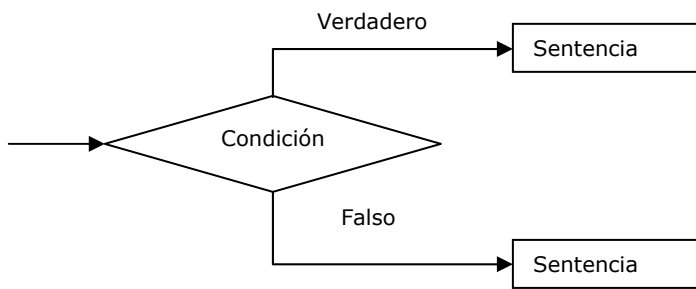
- Entrada o salida por fichas (actualmente en desuso).
- Salida por impresora.
- Entrada o salida por disco magnético.
- Entrada o salida en cinta (su uso es escaso).
- Entrada por teclado.
- Llamada a otro módulo (subprograma o función).

Una vez entendidos los diagramas anteriores, se definirán ahora las tres estructuras básicas dichas por Bohm y Jacopini mediante estos diagramas:

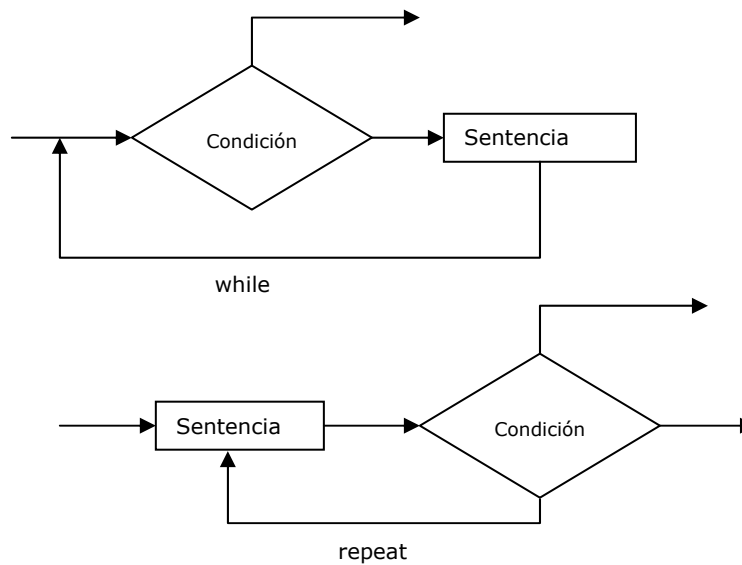
Secuencial:



If – then – else:



Bucle:



Después de haber explicado los diagramas de Chapin y haber definido las estructuras básicas mediante estos diagramas, cabe decir que estos no son los únicos diagramas que existen para hacer la planificación de un programa, aunque ese tema no es motivo de este estudio.

Como se ha dicho anteriormente, para desarrollar un programa lo primero es la planificación del mismo mediante los diagramas citados anteriormente. Una vez hecho eso, se empieza a programar.

La estructura de un programa en Pascal es la siguiente:

```
program <NombrePrograma>;  
  
uses <Archivos de funciones estándar usadas>;  
  
const <declaración de constantes>;  
  
var <declaración de variables>;  
  
begin  
  <Sentencias>;  
end.
```

- a) **Program** es una palabra reservada de Pascal que sirve para iniciar todos los programas e identifica el nombre del programa. Es de uso superfluo ya que no sirve para realizar ninguna acción.
- b) **Uses** también es una palabra reservada y sirve para hacer uso de las librerías estándar de Pascal que contienen las funciones más comunes usadas por el mismo, como veremos a continuación. Por defecto siempre se incluye la librería “System” donde están la mayoría de las función estándar por lo que a veces no es necesario poner los “Uses”.
- c) **Const** es para la declaración de constantes.
- d) **Var** es para la declaración de variables.
- e) **Begin** es el punto de entrada del programa. La primera sentencia que se ponga después del begin será la primera que se ejecuta y luego en forma secuencial se irán ejecutando el resto de sentencias hasta que se termine el programa. Aquí dentro puede y debe ir cualquiera de las sentencias definidas anteriormente con lo que se conforma el programa.
- f) **End** es el punto de terminación del programa.

La estructura del programa debe ser tal y como se ha descrito, ya que la declaración de constantes o variables o los “uses”, si van dentro del cuerpo del programa (el begin y el end), el compilador dará un error.

Entrada / salida:

Antes de continuar cabe hacerse una pregunta: *¿cómo se interactúa con el programa?*. Esa pregunta tiene fácil respuesta: *Con las funciones estándar de entrada y salida de Pascal*. Estas funciones son funciones sencillas que sirven para sacar mensajes por pantalla y recibir información por teclado. Hay más pero nosotros veremos las dos más usadas:

Write();

Esta función lo que hace es escribir un texto por pantalla, por lo tanto, dentro de los paréntesis hay ponerle un texto. Un ejemplo sería:

```
var S: String;

S := 'Hola esto es un programa.';

Write('Hola mundo. ');
Write(S);
```

La primera función escribiría en la pantalla “Hola mundo.” Y la segunda escribiría “Hola esto es un programa.”. Como hay que pasarle un texto, vale lo mismo que se le pase directamente o que se le pasa un identificador de tipo **String** que contiene un texto.

Esta función también puede escribir valores de tipo entero por pantalla. Para ello se le pasa como parámetro el identificador del entero. Si va acompañado por una cadena, irá entre comas. Esto se ve mejor con un ejemplo:

```
var N: Integer;
```

```
N := 10;

Write(n);
Write('Se va a imprimir el número ',n);
Write('Tengo ',n,' años.');
```

La primera función sólo imprimirá “10”. La segunda función con estos parámetros imprimirá por pantalla “Se va a imprimir el número 10”. La tercera imprimirá “Tengo 10 años.”.

Esta función se complementa con la función **WriteLn()** que, a parte de imprimir un texto por pantalla, añade al final un retorno de carro.

Read():

Mediante la sentencia **Read()**, el Pascal permite la lectura de una variable mediante el teclado y la asignación del valor leído a una variable previamente definida. Para hacerla funcionar, a **Read()** se le pasa como parámetro una variable del tipo del que queremos leer el valor, teniendo en cuenta que todo lo que se le pasa por teclado son caracteres y que si se le pasa un entero, **Read()** hará la conversión de dichos caracteres, pasados por teclado, a entero para poder introducirlos en la variable pasada. Por tanto, un ejemplo sería el siguiente:

```
var N: Integer;
    S: String;

Read(N);
Read(S);

WriteLn('Las variables introducidas son ',N, ' y ',S,'.');
```

El usuario introducirá el valor entero y cuando presione <Enter>, **Read()** actuará un meterá el valor leído en la variable “N”. Si el usuario no introduce un valor entero, se producirá un error. Después de presionar <Enter> se pasará a ejecutar el segundo **Read()** donde hay que introducir una cadena, por tanto aquí no se producirá error con ningún carácter del teclado. Esto se puede simplificar de la siguiente forma:

```
var N: Integer;
    S: String;

Read(N,S);

WriteLn('Las variables introducidas son ',N, ' y ',S,'.');
```

Donde **Read()** introducirá cada valor según el número de veces que se presione la tecla <Enter>. El **Read()** se puede ampliar con **ReadLn()** que funciona exactamente igual pero que una vez leído, va a la siguiente línea de lectura.

Después de la definición de las funciones más usadas, ya se puede recurrir al desarrollo del primer programa en Pascal. Dicho programa pedirá al usuario las medidas de un cuadrado y luego imprimirá su área y su perímetro. El código es el siguiente:

```
program AreaPerimetroCuadrado;

uses System;
```

```

var Lado: Integer;
    Area: Integer;
    Perímetro: Integer;

begin
  WriteLn('Introduzca el lado del cuadrado:');
  Read(Lado);
  Area := Lado * Lado;
  Perímetro := Lado * 4;
  WriteLn('El área es ',Area,' y el perímetro es ', perímetro);
end.

```

Otro ejemplo que utilice algunas de las estructuras de control de flujo puede ser el siguiente: un programa que pida los varios elementos de tipo entero, los meta en un array y luego halle la media aritmética. El código es el siguiente:

```

program MediaAritmetica;

const MaxElem = 5;

type ArrayDatos: array[1..MaxElem] of Integer;

var Datos: ArrayDatos;
    i: Integer;
    Resultado, Sum: Integer;

begin
  for i := 0 to MaxElem do Datos[i] := 0;

  for i := 0 to MaxElem do begin
    WriteLn('Introduzca el elemento ',i,':');
    Read(Datos[i]);
  end;

  Sum := 0;
  for i := 0 to MaxElem do begin
    Sum := Sum + Datos[i];
  end;
  Resultado := Sum div MaxElem;

  WriteLn('La media aritmética es: ',Resultado);
end.

```

En principio se define una constante que es el número de elementos del array (por tanto si hay que cambiar el número de elementos del mismo sólo hay que cambiar el valor de esta constante y, a su vez, se cambia en todos los sitios que es utilizada como en los bucles), luego un tipo nuevo que es el array donde se van a guardar los datos. Las variables, una que es del tipo que antes hemos creado para guardar los datos, una variable genérica “i” para recorrer el bucle, el resultado y la suma de los elementos. Todos ellos de tipo entero. Al comienzo del programa se inicializan todos los elementos del el array a 0 mediante un bucle “for”. Luego, con otro bucle “for” y varias sentencias en su interior, se piden los elementos para los que se quiere hacer la media aritmética. Recordemos que **Read()** actúa cuando se presiona <Enter>, por tanto el bucle esperará hasta que el usuario haya presionado dicha tecla. Seguidamente se inicializa la variable “Sum” a 0 y mediante otro bucle “for” se suma todos los valores del array que se guardan en la variable “Sum”. Finalmente se guarda el resultado de dividir la suma entre el número de elementos en la variable “Resultado” que luego se imprime. Cabe

destacar que la división se hace mediante la palabra reservada “**div**”, que es la división entera en Pascal. Para la división real se usa el signo “/”.

PROCEDIMIENTOS Y FUNCIONES

Imaginemos que hay una parte del programa (una sección de código) que por algún motivo hay que repetir varias veces a lo largo del programa. En programas pequeños, como el que hemos visto anteriormente, no ocurre, pero cuando la aplicación tiene más de 100 líneas (por decir un valor ya que esta cantidad puede oscilar desde unas pocas hasta varios millones) se hace necesario un método para no tener que repetir código, que es bueno para el programador y para el compilador, ya que las repeticiones de código se compilan igual y ocupan espacio y recursos.

El mejor método y el que provee el Pascal es la definición de procedimientos y funciones, que son trozos de código identificados por un nombre que se pueden llamar desde cualquier parte del programa y el número de veces que sea necesario siempre atendiendo a los parámetros que hay que pasarle.

Un procedimiento se define de la siguiente forma:

Procedure <Identificador>(<Parámetro 1: Tipo>, <Parámetro 2: Tipo>, ..., <Parámetro N: Tipo>);

Donde “identificador” es el nombre por que cuál se le llamará a lo largo de toda la aplicación y parámetros son una serie de variables que serán las que usará dicho procedimiento dentro de él.

Otra forma de definir partes de código son las funciones, que son exactamente iguales que los procedimientos con la única diferencia de que estos tienen un valor de retorno. La declaración es la siguiente:

Function <identificador>(<Parámetro 1: Tipo>, <Parámetro 2: Tipo>, ..., <Parámetro N: Tipo>) : <Tipo de dato devuelto>;

Un ejemplo de la declaración y llamada de procedimientos y funciones puede ser el programa siguiente que hace lo mismo que el anterior sin procedimientos ni funciones:

```
program MediaAritmeticaConFunciones;  
  
const MaxElem = 5;  
  
type ArrayDatos: array[1..MaxElem] of Integer;  
  
var DatosMedia: ArrayDatos;  
    Resultado: Integer;
```

```

procedure CogeDatos(var Datos: ArrayDatos);
var i: Integer;
begin
    for i := 0 to MaxElem do Datos[i] := 0;

    for i := 0 to MaxElem do begin
        WriteLn('Introduce el dato ',i);
        Read(Datos[i]);
    end;
end;

function MediaAritmetica(Datos: ArrayDatos): Integer;
var Sum,i:Integer;
begin
    Sum := 0;
    for i := 0 to MaxElem do begin
        Sum := Sum + Datos[i];
    end;
    MediaAritmetica := Sum div MaxElem;
end;

begin
    CogeDatos(DatosMedia);
    Resultado := MediaAritmetica(DatosMedia);
    WriteLn('El resultado con funciones es ',Resultado);
end.

```

La primera pregunta que se plantea uno después de ver este programa es ¿por qué se pone la palabra “**var**” dentro de los parámetros del procedimiento?. La respuesta es sencilla aunque requiere una explicación previa:

Hay dos tipos de paso de variables, los que se llaman por **valor** o por **referencia**, cuando es por valor, en los parámetros del procedimiento no lleva ninguna palabra “var”. Esto es, se pasa sólo el valor de la variable, no la variable en sí, se pasa una copia de la misma, por tanto, si ese valor es modificado dentro de la función, cuando la función o procedimiento termina, el valor primero de la variable no es alterado. Por ejemplo, si a una función se le pasa una variable por **valor** de tipo entero con valor 10 y dentro del procedimiento ésta se modifica el valor 28, cuando el procedimiento acaba, la variable sigue valiendo 10.

En cambio, cuando una variable se pasa por referencia (se pone la palabra “var” delante de la variable que queremos pasar por referencia), lo que se pasa en realidad es la dirección de memoria de la misma, por tanto, todas las modificaciones que se realicen sobre esa variable dentro de dicho procedimiento, cuando se termine la función, la variable seguirá cambiada. No se pasa una copia del valor de la variable sino que se pasa una referencia a la misma. Por ejemplo, con la variable de antes que valía 10, la pasamos por referencia y dentro del procedimiento pasa a valer 28, cuando la función acabe la variable valdrá 28.

Después de esta explicación pequeña pero no por ello poco importante, hay que diferenciar entre dos tipos de variables (y constantes): locales y globales.

Las variables locales son las que se declaran dentro de un procedimiento o función y sólo están visibles para ese procedimiento o función. Si otra función intenta usar una variable local de otra, el compilador dará un error. Estas variables son

dinámicas, se crean y se destruyen con cada llamada al procedimiento, por tanto, aunque haya muchas, la ocupación de memoria no será importante.

Por otro lado están las variables globales, que se declaran al comienzo o al final del programa pero fuera de cualquier función o procedimiento. Estas variables están visibles para cualquier función o procedimiento de todo el programa, por tanto hay que tener mucho cuidado al usarlas, ya que se pueden sobrescribir sin querer y perder el resultado anterior sin haberlo usado.

Hay que decir que una variable local no puede tomar el mismo nombre que una global, ya que a la hora de usarlas dentro del procedimiento del cual ha sido declarada la variable local, habrá ambigüedad a la hora de identificarlas.

Otra de las preguntas que se deberían hacer sería ¿cómo se devuelve el dato de la función y cómo se guarda el mismo?. La respuesta es que el dato a devolver dentro de la función se asigna al nombre de la función, así se sabe cuando lo queremos devolver. Generalmente la sentencia de asignación de valor de retorno es la última dentro de la función. Y el dato se guarda tomando la función como una variable y haciendo la asignación de la misma a otra variable, la variable donde queremos guardar el valor de retorno. Como ejemplo se puede decir que las dos últimas líneas del programa anterior se pueden poner de la siguiente forma:

```
WriteLn('El resultado con funciones es ',MediaAritmetica(DatosMedia));
```

La función que calcula la media se llama dentro de otra función que es “WriteLn”. Esto se llama pasar una función como parámetro a otra función y es posible ya que la función “MediaAritmetica” devuelve un entero que es lo que usa la función “WriteLn” para imprimir por pantalla el valor.

ARCHIVOS

Una vez explicados todos los aspectos más importantes para el desarrollo de programas en Pascal, es necesario saber como se puede guardar la información que el usuario provee al programa en forma de datos a tratar, para que en posteriores utilizaciones no sea necesaria la introducción de dichos datos de nuevo por parte del usuario. Para ello, el Pascal provee de una serie de funciones y procedimientos estándar para el tratamiento de ficheros. A continuación se definen dichas funciones y para qué sirven, pero antes como se declara un archivo:

```
var f: File;  
    f2: File of Integer;  
    f3: File of Char;  
    t: Text;  
    S: File of DatosArray;
```

Para declarar una archivo en Pascal se hace de las formas anteriormente citadas. En el primer ejemplo se declara un archivo genérico que sirve para todo. En los siguientes casos, y debido a que el Pascal es un lenguaje de programación muy tipado, los archivos se declaran de un tipo determinado para acceder a ellos de una forma sencilla. Para hacer más fácil el tratamiento de archivos de texto, el Pascal tiene definido un tipo de fichero llamado “**Text**” que es los mismo que “**File of Char**”, por tanto, los ejemplos 3 y 4 son el mismo.

Las funciones de acceso a archivos, una vez declarados de la forma anterior, son las siguientes:

procedure Assign(var F: File; Name: String);

Asigna a una variable de tipo archivo el nombre de un archivo externo. Esto es necesario para que el resto de funciones funciona correctamente. Name es la ruta completa del archivo a tratar.

procedure Reset(var F: File);

Abre un archivo existente. Después de usar la función anterior se usa esta para abrir el archivo y poder trabajar con él.

procedure Rewrite(var F: File);

Crea y abre un archivo. Crea el archivo asignado mediante **Assign()** y luego lo abre, por tanto no es necesario utilizar **Reset()**.

procedure Read(F: File; var V: Tipo);

Lee de la posición actual de un archivo F un valor que es asignado a la variable V del tipo que sea. Esta variable y el tipo del archivo han de ser el mismo. El archivo debe estar abierto. En cada lectura de dato, el puntero de posición va a la posición siguiente (no es necesario usar **Seek()**).

procedure Write(F: File; V: Tipo);

Escribe en la posición actual de un archivo F la variable del tipo indicado. El tipo de la variable y el tipo del archivo deben ser el mismo. El archivo debe estar abierto. En cada escritura va a la posición siguiente (no es necesario usar **Seek()**).

procedure Seek(var F: File; Position: LongInt);

Va a la posición “Position” del archivo indicado por la variable “F”. Hay que tener en cuenta que las posiciones no son por bits ni por bytes sino que son por tamaño del dato de cuyo tipo es el archivo. Si el archivo es de Integer, **Seek()** irá a la posición indicada a nivel de entero. Si es del tipo de una estructura, **Seek()** irá a la posición indicada a nivel de estructura.

procedure Close(var F: File);

Cierra un archivo abierto. Esto es necesario para que los cambios efectuados en el archivo se actualicen en el dispositivo de almacenamiento.

function Eof(F: File): Boolean;

Devuelve True si la posición actual del fichero F es la última (End Of File), sino, devuelve False.

function Eoln(F: File): Boolean;

Devuelve True si la posición actual del archivo es un fin de línea (End Of LiNe), sino, devuelve False.

Ahora veremos un ejemplo de estas funciones con un programa que coge un archivo de texto que se le pasa como parámetro a dicho programa e imprime todo su contenido (volcado de fichero). (La recuperación de parámetros pasados al programa se hace mediante la función “ParamStr(NumParam)”):

```
program VuelcaFicheroTexto;
uses Crt;
var F: Text;
    Ch: Char;
begin
  Assign(F,ParamStr(1));
  Reset(F);
  while not Eof(F) do begin
    Read(F,Ch);
    Write(Ch);
  end;
  Close(F);
end.
```

El tratamiento del resto de tipo de archivo es de la misma forma. La única diferencia es que en lugar de declarar el archivo de tipo texto y la variable de lectura de tipo char, se declaran del tipo necesario. Si es de enteros con Integer, si es de una estructura con ese tipo de estructura, etc. Para acceder a las estructuras leídas, como a las estructuras normales.

PUNTEROS

Se define como puntero (o apuntador según algunos autores) como una variable cuyo valor es la dirección de memoria de otra variable. Esto es, si tengo una variable de entero llamada N, puedo crear una variable de tipo puntero llamada NP cuyo valor sea la dirección de memoria de la variable N.

¿Y para qué se usa esto?. El mayor uso de los punteros es la utilización de memoria dinámica ya que todos los tipos que hemos visto hasta este momento son estáticos. Como ejemplo poner el de un array, donde, en el caso de querer insertar un elemento entre medias, todos los elementos a la derecha del elemento a insertar deberían ser movidos. Y también el tamaño del array es estático, se define en tiempo de compilación.

Para solventar estos problemas, y algunos otros, se usan los punteros, cuya definición es la siguiente:

```
var P: ^Integer;
```

o también y de una forma más útil:

```
type PunteroAEntero = ^Integer;  
var P: PunteroAEntero;
```

Como se ha dicho antes, los punteros se pueden usar para hacer uso de la memoria dinámica, y el mayor y extendido ejemplo es el de la lista lineal enlazada, que es una lista de registros donde uno de los campos es un puntero al registro siguiente, por tanto, las operaciones de mantenimiento de dicha lista (como insertar, borrar, etc.) son muy sencillas de implementar. Aquí se muestra un ejemplo de dicha lista. Supongamos que hay un archivo del tipo del registro siguiente:

```
program ListaEnlazada;  
  
type PunteroSig = ^NodoLista;  
     NodoLista = record  
                   Cadena: String;  
                   Valor: Integer;  
                   Siguiente: PunteroSig;  
                 end;  
  
var Principio: PunteroSig;  
     F: File of NodoLista;  
  
procedure CargaLista(var P: PunteroSig);  
var Aux: PunteroSig;  
begin  
  Assign(F, ParamStr(1));  
  Reset(F);  
  
  New(Aux);  
  P := Aux;  
  while not Eof(F) do begin  
    Read(F, Aux);  
    New(Aux^.Siguiente);  
    Aux := Aux^.Siguiente;  
  end;  
  Aux := nil;  
  
  Close(F);  
end;  
  
begin  
  CargaLista(Principio);  
  while Principio <> nil do begin  
    WriteLn('Cadena: ', Principio^.Cadena, ' - Valor: ', Principio^.Valor);  
    Principio := Principio^.Siguiente;  
  end;  
end.
```

Después de ver este programa cabe decir que el lenguaje Pascal no es uno de los mejores lenguajes para el tratamiento de punteros.

A parte de las listas enlazadas se pueden hacer listas doblemente enlazadas (un enlace al elemento siguiente y otro hacia el elemento anterior), listas circulares (el primero enlazado con el último), listas triplemente enlazadas (enlazadas con el elemento

anterior, con el siguiente y con el siguiente del siguiente), pilas (en lugar de hacerlas con arrays hacerlas con punteros), etc.

El uso de punteros es más amplio de lo que aquí se va a ver, sobre todo para el uso de memoria dinámica y para la asignación de direcciones de memoria absolutas para el manejo de dispositivos externos (puertos paralelo, PCI, ISA, etc.).

APÉNDICE A - PALABRAS RESERVADAS EN PASCAL

Una palabra reservada de un lenguaje de programación es una palabra que el lenguaje reserva para sí por tener un sentido muy específico. Las palabras reservadas no se pueden usar para otra cosa más que para las que vienen definidas. Todas las palabras reservadas anteriormente citadas en el código incluido vienen en negrita. A continuación se muestra un resumen de las mismas en Pascal:

```
and
array
begin
case
const
div
do
downto
else
end
file
for
```

```
function
goto
if
in
label
mod
nil
not
of
or
packed
procedure
```

```
program
record
repeat
set
then
to
type
until
var
while
with
```

A parte de las palabras reservadas hay también una serie de palabras predefinidas que son unas funciones dentro del Pascal para facilitar la programación. Entre ellas se han visto **Read()**, **Write()**, **Reset()**, **Close**, etc.

También hay una serie de caracteres especiales que no se pueden usar para otra cosa. Son estos:

+ : suma.
- : resta.
* : multiplicación.
/ : división real.
= : igual.
< : menor que.
> : mayor que.
<= : menor o igual.
>= : mayor o igual.
<> : distinto.
() : paréntesis.

[] : corchetes.
(* *) : símbolo para comentario.
{ } : llaves (comentarios).
@ : arroba (dirección de memoria).
^ : puntero.
:= : asignación.
. : punto.
, : coma.
; : punto y coma (fin de sentencia).
: : dos puntos.
' : comilla simple.

APÉNDICE B - FUNCIONES Y PROCEDIMIENTOS PREDEFINIDOS

- `abs(x)` : devuelve el valor absoluto de `x`.
- `sqr(x)` : devuelve el cuadrado de `x`.
- `arctan(x)` : devuelve el valor del arco tangente de `x` en radianes.
- `cos(x)` : da el coseno de `x`.
- `sin(x)` : da el seno de `x`.
- `exp(x)` : da el valor de `e` elevado a `x`.
- `ln(x)` : devuelve el logaritmo neperiano de `x`.
- `sqrt(x)` : da la raíz cuadrada de `x`.
- `odd(x)` : devuelve `True` si `x` es impar.
- `trunc(x)` : trunca `x` real a su valor entero.
- `round(x)` : redondea `x` real a su valor entero.
- `ord(x)` : halla el valor ordinal del tipo ordinal `x`.
- `chr(x)` : para `x` entero, entrega el carácter cuyo ordinal es `x`.
- `succ(x)` : da, para cualquier tipo ordinal, el valor del sucesor.
- `pred(x)` : da, para cualquier tipo ordinal, el valor del predecesor.
- `new(Puntero)` : reserva memoria para `Puntero`.
- `dispose(Puntero)` : libera la memoria de `Puntero`.

Curso desarrollado por:

Diego Lago González
Estudiante de Informática
Universidad de León

Héctor López Fernández
Estudiante de Informática
Universidad de León

Curso desarrollado para:

Unicyber

Bibliografía:

“Programación con el lenguaje Pascal”

Editorial Paraninfo

Autores: Francisco J. Sanchís Llorca y Ángel Morales Lozano

Quinta Edición

ISBN: 84-283-1121-8