Sintaxis y Semántica del Lenguaje
El lenguaje Pascal

Este apunte resume las componentes básicas del lenguaje Pascal. Para obtener mayor información se recomienda consultar la siguiente bibliografía:

-. Programación en TURBO Pascal Versiones 5.5, 6.0, 7.0. L. Joyanes Aguilar, Segunda edición. Ed. McGraw Hill
-. Intermediate problem solving. Segunda edición. Helman - Veroff

Introducción

Pascal es un lenguaje que permite programar en forma estructurada y modularizada. Esto significa que es posible dividir al programa en módulos (implementados utilizando procedimientos y funciones) y organizado de manera tal que se pueda leer con facilidad. Esto último tiene que ver con una buena indentación, comentarios, nombres mnemotécnicos, etc. Aunque estas últimas características NO son imprescindibles en un programa Pascal, en la cátedra se exigirá su cumplimiento.

Tipos

El lenguaje Pascal es tipado, lo que significa que casi[1] todas sus variables deben tener un tipo asignado explícitamente.

El tipo de una variable determina el rango de valores que la misma puede contener y el conjunto de operaciones que se pueden aplicar sobre ella. En Pascal existen tipos predefinidos (provistos por el lenguaje) y los definidos por el usuario.

Tipos predefinidos

Los tipos predefinidos son:

- Tipo entero
- Tipo real
- Tipo carácter
- Tipo lógico

[1] Existen algunas excepciones que no se van a tratar aquí.

Tipo entero

Una variable de tipo entero (integer) ocupa, por lo general, 2 bytes[2] de memoria y el rango de valores va desde -32768 a 32767. En realidad, existen otros 4 tipos de datos enteros, con diferentes rangos de valores:

tipo	Rango	#bytes
byte	0..255	1
shortint	-128 ..127	1
integer	-32768..32767	2
word	0..65535	2
longint	-2147483648.. 214748364	4

Tipo real

El tipo real (real) está definido como punto flotante, es decir, fracción*exponente. Turbo Pascal diferencia 3 tipos de flotantes:

Tipo	#bytes
simple	4 bytes
doble	8 bytes
extendido	10 bytes

Tipo Carácter

Hay dos tipos de caracteres:

- Char: un único carácter
- String: un conjunto de caracteres

Este tipo de datos identifica letras, números, signos de puntuación y todos aquellos símbolos que aparecen en la tabla ASCII.

Ejemplo:

'a': el carácter a

'la cadena': una cadena de caracteres

1: el entero 1

'1': el carácter 1

El tipo string define cadenas de 255 caracteres (si pensamos que un carácter en la mayoría de las máquinas se almacena en un byte, entonces un string ocupará 255 bytes). También es posible definir strings más pequeños que 255.

Ejemplo:

string; → define una cadena de 255 caracteres

string[20]: → define una cadena de sólo 20 caracteres

[2] Esto depende también de la máquina / plataforma sobre la cual se está trabajando.

Tipo lógico

En Pascal el tipo lógico (boolean) admite únicamente dos valores: true y false. Cualquier expresión lógica retorna valores de este tipo: verdadero o falso.

Operaciones

Lo que se vio hasta ahora es la estructura que tiene los tipos predefinidos y el rango permitido de valores para cada uno, ahora se verá cuáles son las operaciones permitidas sobre cada tipo:

- OPERACIONES DE RELACIÓN: En todos los casos es posible realizar las operaciones de comparación por menor, mayor, igualdad y desigualdad. (<, <=, >, >=, =, ¬= o <>)

- OPERACIONES ARITMÉTICAS: Con los datos numéricos (integer y real) es posible realizar las operaciones aritméticas: +, -, /, *.

- OPERACIONES LÓGICAS: Con los datos lógicos además están los operadores AND, OR, NOT.

- OPERACIONES DE CONCATENACIÓN: Sólo para los caracteres, y está representada por el signo +.

Subrangos

A veces se necesitan variables que sólo tomarán cierto rango de valores y no todo el permitido por el tipo. En estos casos, es posible acotar estos valores especificando sólo el valor más pequeño y el más grande que dichas variables pueden tomar:

constanteMenor.. constanteMayor

Ejemplo:

'A'..'Z';
1..100;

Tipos definidos por el usuario

Muchas veces es necesario manejar información cuyo tipo no está provisto por el lenguaje. En este caso, pascal permite que el programador defina nuevos tipos y los utilice luego. Un nuevo tipo se define utilizando la cláusula **type**.

Type nombreTipo = tipoExistente;

Ejemplo:

```
Type Edad = Integer;
Type OtraEdad = 0..110;
Type Cadena = string[30];
```

Los nombres de los tipos son indentificadores y, por lo tanto siguen las reglas para nombrar identificadores que se detallan a continuación.

Identificadores

Los identificadores en Pascal comienzan siempre con una letra (a..z,A..Z) o con el símbolo _. y el resto pueden ser cualquier letra, _ y/o dígitos. Con un identificador no solo hacemos referencia a un nombre de variable, sino que también a nombre de tipos, procedimientos, funciones, etc.

Las letras mayúsculas y minúsculas se consideran sin distinción y pueden tener cualquier longitud.

Declaración de variables

Todos los identificadores que se utilicen en un programa Pascal se **DEBEN** declarar **ANTES** de su uso.

Las variables se declaran utilizando la sentencia **var.**

var

x:integer → define una variable denominada x de tipo integer.

y: boolean → define una variable denominada y de tipo boolean.

z: 0..110 → define una variable denominada z de tipo integer que sólo puede tomar los valores de 0 a 110.

E: Edad → define una variable denominada e del tipo Edad (definido anteriormente)

E1:otraEdad → define una variable denominada e1 del tipo otraEdad (definido anteriormente)

Asignación en Pascal

La asignación es el mecanismo por el cual se liga una variable a un valor determinado. En Pascal la asignación se representa por la operación ":=".

Ejemplo:

```
a := 10;
cad := 'esta es una cadena';
```

Estructuras de Control en Pascal

En Pascal existen sentencias simples y compuestas. Las sentencias compuestas se encierran entre las palabras claves begin-end.

En esta sección se detallarán cómo se definen en Pascal las estructuras de control.

Sentencias Condicionales

Permiten realizar selecciones en el flujo de control del programa. Hay dos tipos de selecciones:

- Selecciones simples (if)
- Selecciones múltiples (case)

Sentencias if

Su forma general es:

if expresión
then sentencia1
else sentencia2

donde:

expresión es una expresión lógica que retorna true o false
sentencia1 y sentencia2 son sentencias Pascal simples o compuestas.

DESCRIPCIÓN:

sentencia1 se ejecuta sólo si expresión es verdadera, en caso contrario se ejecutará sentencia2

NOTA: el else es opcional y NO hay cierre de la sentencia;

Ejemplo:

```
1)  if num>10 then a:=a+1
        else a:=a-1;

2)  if num>10 then x:=x+1;
```

Sentencia Case

Su forma general es:

case expresión
Valor**:** sentencias
Valor1, valor2**:** sentencias
Valor3..valor4**:** sentencias;

Valor 5..valor6, valor7**:** sentencias
else sentencias
end;

DESCRIPCIÓN:

Permite elegir entre varias alternativas sin tener que anidar muchas sentencias if. Consiste de una expresión que puede ser de cualquier tipo ordinal (con orden) y una lista de sentencias cada una precedidas por uno o más valores del mismo tipo que la expresión. De acuerdo al valor de la expresión se ejecutará aquella sentencia que coincida con dicho valor. Si ninguna de las sentencias coincide, no se ejecuta ninguna y se sigue en la instrucción siguiente al fin del case. Existe una cláusula opcional, el else, con la cual, se generalizan todos los valores no discriminados en el case:

NOTA: el else es opcional y en este caso se cierra con el end final.

Ejemplo:

1)
```
case sexo of
    'f', 'F': mujeres:= mujeres +1
    'm', 'M': varones := varones +1
end;
```

2)
```
case edad of
    0..2: bebes := bebes +1
    3..11: ninios:= ninios +1
    12..17: adolescentes := adolescentes + 1
    else: adultos:=adultos+1
end;
```

Sentencias de Repetición

Estas sentencias permiten iterar o repetir una o más sentencias. Existen tres diferentes sentencias:

- while
- repeat until
- for

Sentencia while

Su forma general es:

while expresión **do**
    sentencia { puede ser simple o compuesta }

DESCRIPCIÓN:

sentencia se ejecuta siempre que expresión sea verdadera. Cuando es falsa, la iteración termina y continua en la instrucción siguiente

Ejemplo:

```
.....
while num>0
    do num:= num-1;
.....
```

Sentencia repeat until

Su forma general es:

```
repeat
    sentencia1
    sentencia2
    .....
until expresión
```

DESCRIPCIÓN:

Antes de evaluar la expresión se ejecutan las sentencias encerradas en la instrucción y a continuación se evalúa la expresión para determinar su valor de verdad. Si el verdadera, se vuelven a ejecutar las sentencias, y si es falsa, la iteración termina y continua en la instrucción siguiente

Ejemplo:

```
.....
repeat
    num:= num-1;
until num<=0
.....
```

Existen tres grandes diferencias con el while:
1) el repeat until permite ejecutar las sentencias por lo menos 1 vez. El while, saltea las sentencias si la expresión es falsa.
2) repeat until ejecuta hasta que la expresión es verdadera y el while ejecuta mientras que sea verdadera. Por lo que son expresiones opuestas.
3) Con la sentencia repeat es posible usar varias sentencias sin usar una sentencia compuesta, mientras que en el while, es necesario usar begin..end.

Sentencia for

Su forma general es:

for índice**:=** exp1 **to** exp2
do sentencia { puede ser simple o compuesta}

donde:

la variable índice es de algún tipo escalar (entero, char, boolean o enumerativo) y exp1 y exp2 deben ser del mismo tipo que índice. La variable índice se incrementa de a 1 (uno) después de cada ejecución del for. Se puede decrementar con **downto** en vez de **to.**

DESCRIPCIÓN:

Básicamente se ejecuta un conjunto de sentencias un número finito de veces, mientras que una variable (el índice o variable de control) pasa a través de un rango de valores

Ejemplo:

```
.....
for i:=1 to 10 do x:=x+1;
.....
```

Estructura de un programa Pascal

Un programa Pascal está formado por los tipos, variables, procedimientos y funciones que se necesitan para dicho programa. Todo objeto referenciado en el programa **debe** estar declarado con anterioridad. El orden en el cual se declaran los identificadores, no necesariamente debe ser el siguiente pero la cátedra lo recomienda.

```
program nombre;
{ para trabajar con unidades compiladas en forma separada}
uses
{ definición de constantes}
const
{ declaración de tipos definidos por el usuario}
type
{ declaración de variables}
var
{ definición de procedimientos y funciones}

begin
{ Cuerpo del programa}
end.
```

Procedimientos y funciones

Una de las formas de encarar un problema es descomponerlo en problemas o módulos más pequeños y más sencillos de implementar. En Pascal, estos módulos se implementan mediante procedimientos y funciones. Tanto un procedimiento como una función son subprogramas que realizan alguna tarea especifica.

Las diferencias entre ellos son:

- Las funciones devuelven un **único** valor a la unidad que la llamo (el programa principal o un procedimiento o función) En cambio los procedimientos pueden retornar 0, 1 o más valores.
- Una función se referencia utilizando su nombre en una expresión, mientras que un procedimiento se referencia por una llamada o invocación.

Procedimientos

La forma general de declararlos es

```
procedure nombreProc (lista de parámetros formales)
        { declaraciones locales }
begin
        { cuerpo del procedimiento }
end;
```

Se lo invoca en el programa principal o en otros subprogramas por su nombre:

```
NombreProc;
```

ó

```
nombreProc(parámetros reales);
```

Cuando se lo invoca, se localiza el procedimiento y se ejecutan las instrucciones del cuerpo, luego el control vuelve a la instrucción siguiente de donde se lo invoco.

Ejemplo:

```
program demo;
...
procedure A;
        begin
        end;
....
begin
......
A;
.....
end.
```

Definición del procedimiento A

Invocación del procedimiento A

¿Que ventajas tiene la utilización de procedimientos?

- Ahorro de líneas de código: no repetir código en el mismo programa
- Reusabilidad: utilizar el mismo código en distintos programas. Utilizarlos como un caja negra que acepta determinadas entradas y produce determinadas salidas.
- Modularidad: se divide un problema en tareas o módulos más pequeños que pueden ser testados en forma separada.

Parámetros

Marcan la comunicación entre los procedimiento. Debe existir una correspondencia entre los parámetros reales y formales.(en numero, orden y tipo)

Ejemplo:

```
Program demoParametros;
        procedure UNO(a: integer, b:real)
        begin
        end;
var
        x: integer;
        y: real;
begin
        UNO(x);             { incorrecta }
        UNO(y);             { incorrecta }
        UNO(y,x);           { incorrecta }
        UNO(x,y)            { correcta }
end.
```

Pascal posee dos tipos de pasaje de parámetros: por valor o por referencia

Por valor: En la llamada al procedimiento el valor del parámetro real se copia en el parámetro formal y si el parámetro forma se modifica el parámetro real no.
Por referencia: lo que se pasa es la dirección del parámetro real, por lo tanto cualquier modificación del parámetro formal quedara reflejado en el parámetro real.

Ejemplo:

```
Program demoReferencia;
        procedure DOS(var a:integer)
        begin
                a:= a+1;
        end;
        var x:integer;
        begin
                x:=4;
                DOS(x);
        end;
```

En la teoría se explicarán detalladamente las diferencias.

Otro ejemplo de pasaje de parámetros:

```
program OtroEjemplo;
        var a, b : integer;
        procedure Cambio(x,y : integer)
        var
                z:integer;
        begin
                z:=x;
                x:=y;
                y:=z;
        end;
        begin
                writeln('Ingrese dos números');
                readln(a,b);
                Cambiar(a,b);
                writeln('El primer numero que ingreso fue',a);
                writeln('El segundo numero fue',b);
        end.
```

Si este programa intenta intercambiar los valores leídos por teclado, lo hace realmente? Corregirlo.

¿Cuándo se debe utilizar parámetros por valor y cuándo por referencia?

- Si la información que se pasa al procedimiento no tiene que ser devuelta fuera del procedimiento, el parámetro formal que representa la información **puede** ser un parámetro valor (parámetro de entrada)
- Si se tiene que devolver información al programa llamador, el parámetro formal que representa esa información **debe** ser un parámetro por referencia (parámetro salida)
- Si la información que se pasa al procedimiento puede ser modificada y se devuelve un nuevo valor, el parámetro formal que representa esa información **debe** ser un parámetro por referencia (parámetro entrada \ salida)

Funciones

Existen funciones predefinidas por el lenguaje que se pueden utilizar en expresiones: Ord(), Abs(), Sin(), etc.

El usuario puede definir sus propias funciones de igual forma que los procedimientos.

```
function nombre(parámetros): tipo
{ declaraciones locales}
begin
        cuerpo de la función
        nombre:= valor de la función
end;
```

El nombre de la función se refiere a la posición de memoria que contiene el valor devuelto por la misma.

El tipo de datos del resultado debe coincidir con el tipo declarado en la cabecera de la función.

La asignación ultima siempre debe existir.

No debe realizar ninguna otra tarea que calcular el valor del tipo del nombre. (para no tener efectos laterales)

Ej:

```
function triple (num : real) : real
begin
        triple := 3*num;
end;
```

Variables locales y globales

Las variables que intervienen en un programa con procedimientos pueden ser de dos tipos: locales y globales.

Una variable local es aquella que está declarada dentro de un subprograma y se dice que es local al subprograma. Una variable local sólo está disponible durante el funcionamiento del mismo.

Las variables declaradas en el programa principal se denominan variables globales. A diferencia de las variables locales cuyos valores se pueden utilizar sólo dentro del subprograma en que están declaradas, las variables globales pueden ser utilizadas en el programa principal y en todos los subprogramas declarados en él.

```
program variablesLocYGlob;
        var x,y:integer;
        procedure A(x:integer)
        var
                z:integer;
        begin
                ……..
        end;
        begin
                .....
        end.
```

En el caso de que coincidan en un mismo programa, dos variable con un mismo nombre, una global y otra local, la variable local enmascara a la global.

Efectos laterales

Pascal permite a los subprogramas (procedimiento y funciones) modificar el valor de las variables globales que no son parámetros reales o actuales.

Ejemplo:

```
.....
var m:integer;
procedure MalaPráctica;
begin
        writeln('Introduzca un numero');
        readln(m);
end;
.....
```

Este ejemplo no da ningún error de compilación ni en ejecución. En otras palabras, esto está permitido en Pascal. El valor de m se modifica dentro del procedimiento y, por consiguiente si a partir de ahí se utiliza en el programa llamador, tomará ese último valor.

Un **efecto lateral** es la acción de modificar el valor de una variable global en un subprograma.

No se recomienda que un programa produzca efectos laterales. ¿Por qué?
Ejemplo: (Pensar qué sucede en el siguiente ejemplo)

```
program EfectosLaterales;
var
        j:integer;

procedure Lateral;
begin
        for j:= 1 to 100
                do writeln('J ahora vale', j);
end;

begin
        j:= 8;
        writeln(j);       { se visualiza 8 }
        Lateral;
        writeln(j);       { se visualiza 100 }
end.
```

La interacción o comunicación entre un procedimiento / función y el programa (o subprograma) debe realizarse **completamente** a través de los parámetros y **no** de variables globales.

Ámbito o alcance de un identificador

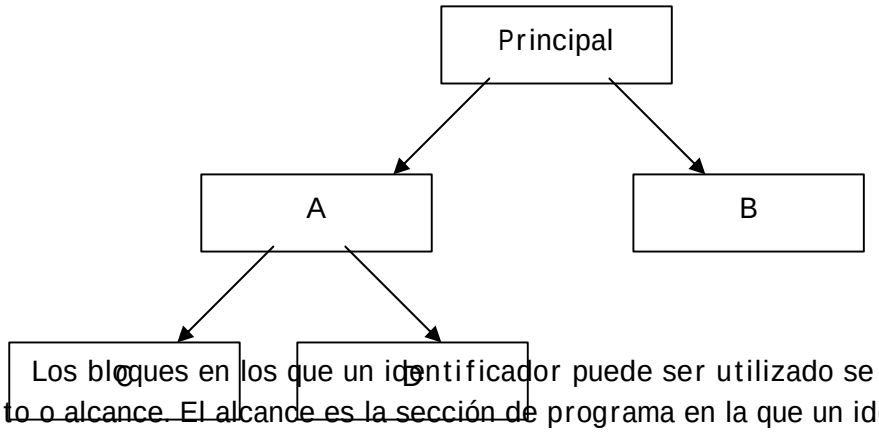
Los programas Pascal tienen estructura tipo árbol. El programa principal es la raíz y de éste penden muchas ramas (procedimiento y funciones).
Ejemplo:

```
program Principal;
        procedure A;
                procedure C;
                begin
                end;
                procedure D;
                begin
                end;
        begin
        end;
        procedure B;
        begin
        end;
begin
end.
```

Un programa puede contener diferentes bloques (procedimiento y funciones) y un procedimiento puede, a su vez contiene otros procedimientos anidados.

La organización de los procedimientos se representa también como bloques de programa. Un bloque es la parte de declaración y el cuerpo de programa o procedimiento.

De esta manera, el programa anterior puede verse como:



Los bloques en los que un identificador puede ser utilizado se conocen como ámbito o alcance. El alcance es la sección de programa en la que un identificador es válido, y donde puede ser referenciado.

Reglas de alcance de Pascal:

1.- Un identificador declarado en un bloque P, puede ser utilizado en P y en todos los procedimientos y funciones anidados en él.

2.- Si un identificador j declarado en el procedimiento P se redeclara en algún procedimiento interno Q (encerrado en P), entonces el procedimiento Q y todos los encerrados en él no pertenecen al alcance de j.

Ejemplo:

```
program EjemploAnidado;
const
        k=2.71828;
var
        x,y:real;

        procedure A (var x:real)
                var
                        m,n:integer;
                procedure B(w:real);
                        var
                                p,q :integer;
                begin
                ...
                end;
        begin
        ....
        end;
        procedure C(var letra:char);
                const
                        IVA= 21;
                var
                        x:char;
        begin
        ...
        end;

begin
...
end.
```

La constante k declarada en el programa principal tiene alcance en todo el programa. La variable x es una variable global pero está redefinida en los procedimientos A y C, por lo tanto tiene alcance sólo en el programa principal y en el procedimiento B.

Ejercicio: Marcar el alcance de todas las variables del ejercicio anterior.

Las reglas de alcance no se definen únicamente para el uso de variables. En ellas se habla de "identificadores" y esto abarca también a los nombres de los procedimientos y funciones.

Ejemplo:

```
program A
        procedure B
                procedure C
                begin
                end;
        begin
        end;

        procedure D;
                procedure E
                begin
                end;
                procedure B
                begin
                end;

        begin
        end;
begin
end.
```

Ejercicio: Marcar los alcances de todos los identificadores.

Arreglos

Un arreglo es un conjunto de elementos de un mismo tipo. Tiene un tamaño fijo que NO puede excederse. Se puede acceder a sus elementos en cualquier orden (acceso directo y aleatorio)

La forma de definirlo es:

```
type arreglo = array[1..7]  of real;

var
        maxTem : arreglo;

ó

var
        otroArreglo : array[1..10]  of integer;
```

Se puede acceder directamente a un elemento del arreglo mediante una expresión llamada índice encerrado entre corchetes.

Ejemplo:

maxTemp[5] : hace referencia al 5º elemento dentro del arreglo maxTemp.

Si k es una variable entera cuyo valor es 5, entonces maxTemp[k] señala el 5º elemento y maxTemp[k+1] el 6º.

Ejercicio: Inicializar el arreglo anterior con un valor determinado

Tipos de los índices

Como vemos, los índices en un arreglo de tipo arreglo deben ser integer (de 1 a 7). En general, los índices deben ser de tipo de datos ordinales (con orden) como los enumerativos, por ejemplo. (los char, integer, etc.)

```
type
    dias = ( DOM, LUN, MAR, MIE, JUE, VIE, SAB);
    arre1 = array [dias] of real;
    arre2 = array['a'..'z'] of integer;

var
    temp. : arre1;
    codigo : arre2;
```

temp, es otro arreglo de 7 elementos reales que NO tienen índice reales (temps[MAR] es el 3º elemento de temp).

Arreglos multidimensionales

Si se quiere generar las temperaturas mínimas de cada día de la semana durante 52 semanas (1 año) se puede utilizar una matriz:

```
type
    matriz = array[1..7,1.52] of real;

var
    tempMin : matriz;    { 7 filas por 52 columnas }
```

tempMin[5,12] indica el elemento de la fila 5 y columna 12

Esto se generaliza a más dimensiones. Aunque en la práctica sólo se utiliza hasta 3 (Tensores).

Arreglos de arreglos

En vez de declara un arreglo e dos dimensiones para las 52 semanas, se puede declarar un arreglo de una sola dimensión de 52 arreglos de 7 elementos.

```
type
        semanas =  array[1..7] of real;
        tempAnuales = array[1..52] of semanas;

var
        temps : tempsAnuales;
```

Aquí, temps[12] es una arreglo de 7 temperaturas correspondiente a las 12º semana. Si pensamos en la información dividida en semanas individuales, esta estructura es más adecuada que la otra. Para referenciar el 5 día de la semana 12:

```
temps[12][5]
```

Ejercicio: Inicializar el arreglo anterior con un valor determinado

Podemos tener array de strings, integer, char, etc.

Registros

Mientras que un arreglo es una colección de elementos del mismo tiempo, un registro es un grupo de ítems relacionados que no necesariamente son del mismo tipo.

Cada ítem del registro se denomina campo.

En forma general un registro en Pascal se define:

```
        record
                campo_1: tipo;
                campo_2: tipo;
                ......
                campo_n: tipo;
        end;
```

```
type
        Alumno = record
                        Nombre : string[30];
                        Legajo : integer;
                        Prom : real;
                        end;
var
        alu : Alumno;
```

Para referenciar un campo dentro del registro se debe calificar utilizando el punto ('.') seguido por el identificador de campo.

Ejemplo:

```
alu.legajo := 12345;
```

Se puede copiar un registro entero a otro.

```
var
    a1, a2 : Alumno;
    .....
    a1 := a2;
```

La sentencia With

Permite referir cada campo con solamente el identificador utilizando la sentencia with.

ej:

```
with  a1 do
    begin
        nombre :=  'Carlos Fernandez';
        legajo := 12345;
        prom := 4.50;
    end;
```

Fuera del with se debe calificar el identificador de campo.

```
writeln(a1.nombre, 'tiene promedio ', a1.prom);
```

NOTA: Si las sentencias dentro del with son muchas, se puede perder legibilidad porque se ven a los identificadores de campo como variables libres y, por lo tanto. se debe restringir el uso a pocas líneas.

Registro dentro de registros

A veces es bueno tener un registro dentro de otro, para independizar datos. Por ejemplo si se quiere agregar al reg. del alumno un campo dirección (calle, numCasa y ciudad)

```
type
    dirección = record
                    calle : string[20];
                    numCasa: integer;
                    ciudad: string[20];
                end;
    Alumno = record
                    nombre : string[20];
                    legajo : integer;
                    dir : dirección;
```

```
            prom : real;
        end;

var
    alu : Alumno;
```

¿Cómo se referencia el campo calle dentro de alumno?

```
    alu.dir.calle
```

Arreglo de registros

Si se quiere tener la información de los estudiantes correspondientes a una clase determinada, se podría definir el siguiente tipo de datos:

```
type
    clase = array (1..90) of Alumno;

var
    clase1: Clase;
```

clase1[9].nombre : es el nombre del 9no. registro.
clase1[9].nombre[1] : es la primera letra del campo nombre del 9 registro.
clase1[9].dir.ciudad : es el campo ciudad del 9no. registro.

Archivos

A diferencia de los otros tipos de datos, los datos guardados en un archivo son permanentes, es decir, siguen existiendo después que el programa termina su ejecución. De esta manera, pueden ser compartidos por varios programas.

Un archivo es una colección de información (datos relacionados entre sí) localizada o almacenada como una unidad en alguna parte de la computadora (memoria externa) A diferencia de los arreglos y registros, el tamaño de esta colección no es fija y está limitado por la cantidad de memoria secundaria (disco o cinta) disponible. Es decir, los archivos son dinámicos, en cambio los registros o arreglos son estructuras estáticas[3].

Los archivos permiten la comunicación entre programas: un programa A puede escribir su salida en un archivo y un programa B puede usarlo como entrada.

[3] Cuando se declara un arreglo se deben especificar el tamaño (los límites) que ocupará esta estructura. Lo mismo sucede con los registros, en donde se detallan cada uno de los campos que lo compone en forma estática. En cambio cuando se declara un archivo no se especifica ningún tamaño. A medida que se van agregando datos, el tamaño se va incrementando dinámicamente (en memoria externa)

Modo de acceso

Existen dos modalidades para acceder a un archivo:

- en forma secuencial: comenzando por el primero, se tratan los elementos en forma secuencial
- en forma aleatoria (acceso directo): se puede acceder a un elemento directamente a través de su posición.

Pascal estándar sólo permite acceso secuencial a los archivos, pero el Turbo Pascal también permite acceso directo.

Archivos en Pascal

Existen tres tipos de archivos en Turbo Pascal:

- Texto (text): con acceso secuencial.
- Tipados (con datos de un tipo específico): con acceso aleatorio
- No tipados (con datos sin un tipo específicos: estos NO se tratarán en la materia.

Mecanismos para manipular archivos

Existen varios procedimientos y funciones predefinidos en Pascal para trabajar con archivos. En este apunte sólo se resumirán brevemente y se recomienda consultar la bibliografía para más información:

Por lo general, la secuencia de operaciones que se realizan con los archivos es la siguiente:

1. Asignación
2. Apertura
3. Tratamiento
4. Cierre

La secuencia anterior es válida tanto para archivos de datos como para archivos de texto, pero las operaciones en cada caso, si bien se denominan de la misma manera su funcionamiento y operación no lo es. Por lo tanto se hará una distinción entre los distintos tipos de archivos: de texto y de datos.

Archivos de Texto

Declaración

En forma general un archivo se declara:

```
type texto:text;
var archi: texto;
        archi1: text;
```

ASIGNACIÓN DE ARCHIVOS

Permite ligar un variable del programa de tipo archivo con un archivo externo (en el disco).

assign (nombreVariableArchivo, nombreExternoArchivo)

nombreVariableArchivo: es el nombre del archivo dentro del programa (el utilizado en su declaración)
nombreExternoArchivo: es el nombre con el que se conoce el archivo por el sistema operativo (ejemplo: c:\sintaxis\trabajos\tp1.txt).

Ejemplo:

```
.....
        var archivotexto: text      → archivo de texto
        begin
                assign(archivotexto, 'c:\sintaxis\trabajos\tp1.txt');
                ....
        end;
.......
```

APERTURA DEL ARCHIVO

Después de ser asignado, el archivo debe ser abierto. Esto es posible utilizando uno de los tres procedimientos predefinidos: reset, rewrite y append.

Reset: abre un archivo existente para lectura.

Rewrite: crea y abre un nuevo archivo. Si el archivo ya existe, borra su contenido.

Append: abre el archivo para escritura, pero **sólo** para añadir al final.

En cualquier caso es posible testear si se produjo algún error en las operaciones anteriores mediante la función IOResult[4].

LECTURA DE UN ARCHIVO

Se refiere a la recuperación de datos del archivo. Las operaciones para leer datos son read y readln[5].

Forma general:

Read (archi, nomVar1, nomVar2, ...):

Lee información del archivo archi y la almacena en las variables nomVar1, nomVar2, etc. Estas variables pueden ser del tipo char, integer, real o string.

Ejemplo:

```
.............
        var archivo: text;
                linea: string[30];
```

[4] Consultar la bibliografía para más detalle.
[5] ReadLn salta al principio de la siguiente línea del archivo.

```
            x: integer;
    begin
            assign(archivo, 'prueba1');
            reset(archivo);
                    .....
            readLn(archivo, linea, x);
                    .....
    end;
.............
```

ESCRITURA DE UN ARCHIVO

Se refiere a la inserción de datos en el archivo. Las operaciones para leer datos son write y writeln[6].

Forma general:

write (archi, nomVar1, nomVar2, ...):

Escribe el contenido de nomVar1, nomVar2, etc. en el archivo archi. Estas variables pueden ser del tipo char, integer, real o string.

Ejemplo:

```
......
    var archivo: text;
            linea: string[30];
            x: integer;
                    .....
            writeln(archivo, linea, x);
                    .....
    end;
.......
```

Aclaración:

El uso de write y writeln en archivos de texto es exactamente el mismo que el usado para mandar información a la pantalla. En realidad, en este caso se utiliza un archivo de texto predefinido denominado output (archivo estándar de salida) que se lo liga a la pantalla por defecto. Lo mismo sucede con la operación read. Existe otro archivo predefinido, denominado input (archivo estándar de entrada) asociado al teclado.

CIERRE DE UN ARCHIVO

La operación para cerrar el archivo es close.

Close (nombreArchivo)

[6] WriteLn salta al principio de la siguiente línea del archivo.

Función EOF()

Es posible preguntar por fin de archivo (símbolo especial EOF) para testear cuando se terminó o finaliza el archivo. Esto es lo que sigue a la última componente del archivo. Esta función retorna el valor true si se encuentra se alcanzó el fin de archivo y false en caso contrario.

Cuando se abre un archivo, un puntero indica la posición del registro actual dentro del mismo. En este caso señalará la posición cero (0). A medida que se realizan operaciones de lectura (o escritura) este puntero avanza a las siguientes posiciones hasta alcanzar la marca de fin de archivo.

0	1	2	3	4	5	EOF

Manejo simple de archivos de texto - Un ejemplo

Supongamos que existe en el disco un archivo denominada entrada.txt y quiero copiar su contenido a otro archivo denominado salida.txt.

```
program copioArchivo;
var
        archiE, archiS: Text;
        linea: string[80];
begin
        assign(archiE, 'c: \sintaxis\entrada.txt');
        assign(archiS, 'c: \sintaxis\salida.txt');
        reset(archiE);
        rewrite(archiS);
        while not eof(archiE)
                do begin
                        readLn(archiE,linea);
                        writeLn(archiS,linea);
                   end;

        close(archiE);
        close(archiS);
end.
```

Archivos de Datos

Declaración

En forma general un archivo de datos se declara:

file of tipoDeDatos

donde tipoDeDatos puede ser cualquier tipo predefinido o definido por el usuario.

Puedo tener archivos de enteros, de caracteres, de registros, etc.

```
var
        arch_Enteros : file of integer;
        arch_Notas : file of char
```

También es posible declarar un tipo archivo:

```
        type
                Item = record
                                nombre,
                                domicilio : string[20];
                                TE : string[15];
                        end;
                AgendaTelefonica = file of Item;
```

LECTURA DE UN ARCHIVO

La operación para leer datos del archivo es read.

Forma general:

Read (archi, nomVar1, nomVar2, ...):

Lee información del archivo archi y la almacena en las variables nomVar1, nomVar2, etc. Estas variables **deben** ser del mismo tipo que el archivo.

Ejemplo:

```
.........
        var archivo : file of integer;
                x, y : integer;
        begin
                assign(archivo, 'prueba2');
                reset(archivo);
                        .....
                read(archivo, x, y);
                        .....
        end;
.........
```

ESCRITURA DE UN ARCHIVO

La operación para escribir datos es write.

Forma general:

write(archi, expresion1, expresion2, ...):

Se almacena la información dada por expresion1, expresion2, etc, en el archivo archi. Las expresiones anteriores **deben** ser del mismo tipo que el archivo.

Ejemplo:

```
var archivo : file of integer;
        x,  y : integer;
begin
        assign(archivo, 'prueba2');
        rewrite(archivo);
                .....
        write(archivo, x, y, 10);
                .....
end;
```

CIERRE DE UN ARCHIVO

La operación para cerrar el archivo es close.

Close (nombreArchivo)

Manejo simple de archivos de datos- Un ejemplo

Este programa lee datos de alumnos de un archivo e imprime la cantidad de alumnos por comisión. Se supone que hay 100 comisiones y los alumnos tienen asignado un número de comisión entre 1 y 100.

```
program promedio_de_alumnos;
const
        MAX_ALUMNOS = 100;
type
        Alumno = record
                        nyap : string[30];
                        nroLegajo : integer;
                        comision : 1..MAX_ALUMNOS;
                end;
        Archi_Alumnos = file of Alumnos;

var
        al : Alumnos;
        archi : Archi_Alumnos;
        comisiones : array [1..MAX_ALUMNOS] of integer;
begin
        Assign(archi, 'c: \sintaxis\datos.dat');
        Reset(archi);
        for i:=1 to MAX_ALUMNOS
                do comisiones[I]:=0;
        while not eof(archi)
                do begin
```

```
                    read(archi,al);
                    comisiones[al.comision]:= comisiones[al.comision] +1;
                end;
        close(archi);
        for i:= 1 to MAX_ALUMNOS
                do if comisiones[I]<>0
                    then writeln('La comisión',i,'tiene ', comisiones[i], ' alumnos')
end.
```

Operaciones para el acceso aleatorio

En Turbo Pascal existe un conjunto de operaciones predefinidas para poder acceder en forma directa a alguna componente del archivo de datos.

PROCEDIMIENTO SEEK: permite posicionar el puntero del archivo en una posición determinada.

Seek (archivo, númeroRegistro)

Suponiendo que el primer registro del archivo es el cero (0) , entonces númeroRegistro es un valor entre 0 y la cantidad de elementos del archivo.

Ejemplo:

Seek (agenda, 5) → mueve el puntero al sexto registro (se comienza desde el 0!!)

FUNCIÓN FILEPOS: Devuelve la posición actual del archivo (número de registro) en forma de un longint. Si está al principio del mismo, devuelve cero (0)

Filepos(archivo);

FUNCIÓN FILESIZE: Devuelve la cantidad de registros del archivo Si el archivo está vacío devuelve cero (0).

Filesize(archivo);

Resumen:

A continuación se detallan la lista de operaciones para manipular archivos en Pascal.

Operación	Descripción	Observación
Assign	Permite ligar la variable dentro del programa con el archivo físico en el	Aplicado tanto a archivos de texto como de datos.

	disco	
Reset	Permite abrir un archivo para lectura/escritura	Aplicado tanto a archivos de texto[7] como de datos
Rewrite	Permite abrir un archivo para escritura	Aplicado tanto a archivos de texto como de datos.
Append	Permite abrir un archivo para escritura (para añadir al final)	Sólo archivos de texto.
Read	Permite leer datos de un archivo	Aplicado tanto a archivos de texto como de datos.
Readln	Idem read, solo que salta a la línea siguiente	Sólo archivos de texto.
Write	Permite grabar datos a un archivo	Aplicado tanto a archivos de texto como de datos.
WriteLn	Idem read, solo que salta a la línea siguiente	Sólo archivos de texto.
Eof	Permite testear el fin de archivo	Aplicado tanto a archivos de texto como de datos.
Eol	Permite testear por fin de línea	Sólo archivos de texto.
Seek	Permite posicionarse en un número de registro determinado	Sólo archivos de datos
Filepos	Permite conocer el número de registro actual	Sólo archivos de datos
Filesize	Permite conocer la cantidad de registros del archivo	Sólo archivos de datos
Close	Cierra un archivo previamente abierto.	Aplicado tanto a archivos de texto como de datos.

Manejo de archivos de datos con acceso directo- Un ejemplo

El siguiente ejemplo se encuentra disponible en el libro de Joyanez Aguilar (pág. 521) en forma completa. En esta sección sólo se hará énfasis en la manipulación del archivo en forma directa.

Enunciado:

Se dispone de un archivo (ya creado) de estudiantes de un instituto. La información que se cuenta en el archivo es la siguiente:

[7] Si es de tipo text solo lectura.

Nombre del alumno, domicilio y edad.

Se quiere realizar un programa que realice altas, bajas, modificaciones y consultas a este archivo.

Implementación en Pascal

```
program accesoAleatorio;
uses Crt;
type
	Cadena = string [40];
	Domicilio= record
			calle	: string [20];
			numero: integer;
			ciudad: string[20];
		end;
	Alumno = record
			nombre: cadena;
			edad		: 0..99;
			dom		:  Domicilio;
		end;
	ArchivoAlumnos = file of Alumno;

	var
		archi : ArchivoAlumnos;
		opcion : char;
procedure LeerRegistro(var alu: Alumno);
begin
	with alu do
		begin
			writeln('Nombre: '); readLn(nombre);
			writeln('Edad: '); readLn(edad);
			writeln(Domicilio - Calle: '); readLn(dom.calle);
			writeln(Domicilio - Número: '); readLn(dom.numero);
			writeln(Domicilio - Ciudad: '); readLn(dom.ciudad);
		end;
end;
function Posicion( nom: cadena; var archi: ArchivoAlumnos) : integer;
var reg: Alumno;
	esta: boolean;
begin
	esta:= false;
	seek(archi, 0);
	while not wof(archi) and not esta
		do begin
			Read(archi, alu)
			esta:= alu.nombre = nom;
```

```
                end;
        if esta
                then Posicion:= filePos(archi) -1
                else Posicion:= -1;
end;

procedure MostrarDatosAlumno(alu: Alumno);
{Muestra en la pantalla los datos del alumno alu}
begin
        ........
end;

procedure Modificar(var archi: ArchivoAlumnos);
var alu: Alumno;
        nom: Cadena;
begin
        Reset(archi);
        WriteLn('Introduzca el nombre del alumno a modificar';
        ReadLn(nom);
        i := Posicion (nom, archi);
        if i= -1
                then writeLn('El alumno buscado NO se está en el archivo')
                else begin
                        seek(archi, i);
                        read(archi, alu);
                        MostrarDatosAlumno(alu);
                        writeLn ('Introduzca los nuevos datos:');
                        LeerRegistro(alu);
                        i := filePos(archi) -1;
                        write(archi, alu);
                        writeLn ('Los datos se han modificado.');
                end;
end;

procedure Consultar(var archi: ArchivoAlumnos);
{Permite consultar los datos de un alumno }
begin
        ......
end;

procedure Eliminar(var archi: ArchivoAlumnos);
{Permite eliminar los datos de un alumno }
begin
        ......
end;
procedure Agregar(var archi: ArchivoAlumnos);
{Permite agregar un nuevo alumno al archivo}
begin
        ......
end;
```

```
begin { Comienzo del programa principal}
        Assign(archi, 'c:\sintaxis\alumnos.dat');
        Repeat
                ClrScr;
                WriteLn('Menú Principal:');
                WriteLn('1.- Agregar Alumno');
                WriteLn('2.- EliminarAlumno');
                WriteLn('3.- Modificar Alumno');
                WriteLn('4.- Consultar Alumno');
                WriteLn('5.- Listado total de alumnos ');
                WriteLn('6.- Salir del sistema');
                WriteLn('Elija una opción:');
                repeat
                        opcion := readKey;
                until opcion >='1' and opcion <='6'
                case opcion of
                        '1':  Agregar(archi);
                        '2':  Eliminar(archi);
                        '3':  Modificar(archi);
                        '4':  Consultar(archi);
                        '5': Listar(archi);
                end;
        until opcion ='6'
end.
```

Punteros

Un puntero es una variable cuyo valor es una dirección de memoria. Nosotros vamos a trabajar con punteros para implementar un tipo de datos que NO se encuentra predefinido en Pascal: tipo Lista.

Declaración de punteros en Pascal:

Se pueden definir tipos punteros o definir variables de tipo puntero:

```
Type
        ptr = ^integer;
Var
        p: ptr;
        q: ^real;
```

La variable p apunta a una variable entera anónima. Es decir que la única forma de acceder a esa variable es a través del puntero.

Ejemplo

p^:= 5 → asigno el valor 5 a la variable apuntada por p.

writeln(p^) → imprime el valor de la variable apuntada por p.

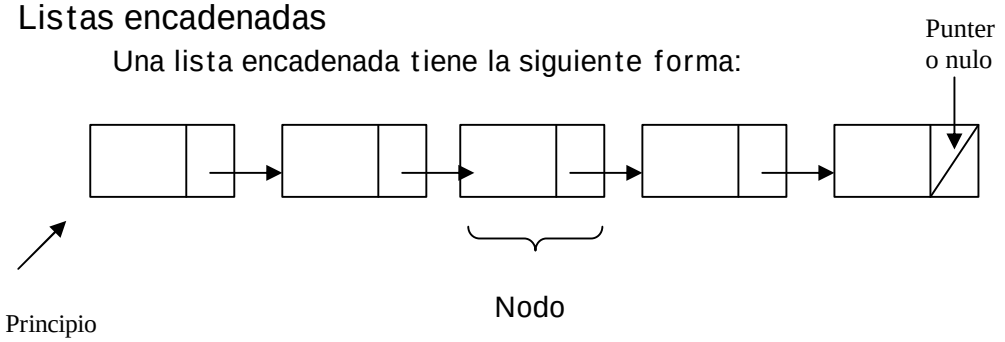
Las variables de tipo puntero apuntan a variables denominadas dinámicas. Estas variables se crean en forma explícita por medio del procedimiento new de Pascal.
new(p) → asigna memoria a la variable apuntada por p.

Ejemplo: (el seguimiento de este program se realizará en a teoría)

```
program demopunteros;
type
        ptr=^integer;
var
        p,q:ptr;
begin
        new(p);
        p^:= 1;
        new (q);
        q^:=2;
        writeln(p^, q^);
        p^:= q^+1;
        writeln(p^, q^);
        p:=q;
        writeln(p^, q^);
        p^:= 7;
        writeln(p^, q^);
        new(p);
        q:=p;
end.
```

Listas encadenadas

Una lista encadenada tiene la siguiente forma:



Una lista es una sucesión de nodos. Cada nodo tiene un valor y un puntero al siguiente nodo. El último nodo tiene un valor especial denominado **nil** que indica al fin de la lista

Definición de listas en Pascal:

```
Type lista = ^nodo;
        Nodo = record
                        Dato: ...
                Psig:lista
```

End;

Siempre se accede a la lista por su comienzo: el puntero inicial a la misma:

Var pi: lista.

¿Cómo se recorre una lista? El procedimiento básico es el siguiente:

Situarse la principio de la lista
Mientas no se llegue al final de la lista
Do begin
Imprimir el dato del nodo actual
Pasar al siguiente nodo
End;

Implementación en Pascal:

```
Procedure imprimirLista(pi:lista)
Var aux:lista
begin
{Situarse la principio de la lista}
aux:= pi;

{Mientas no se llegue al final de la lista}
while aux <>nil
        do begin
        {Imprimir el dato del nodo actual}
        writeln(aux^dato);
        {Pasar al siguiente nodo}
        aux:=axu^psig;
        end;
end;
```

Operaciones para el manejo de la lista

Insertar al principio:

```
Procedure insertarPpio (var l:lista, elem:integer)
Var nuevo: lista;
Begin
        New(nuevo);
        Nuevo^dato:=elem;
        Nuevo^psig:=l;
        L:=nuevo;
End.
```

Insertar al final:

```
Procedure insertarFinal (var l:lista, elem:integer)
Var nuevo, actual, anterior: lista;
Begin
        New(nuevo);
        Nuevo^dato:=elem;
        Nuevo^psig:=nil;
        actual:=l;
        anterior:=nil;
        while actual<>nil
                do begin
                        anterior:=actual;
                        actual:=actual^psig;
                        end;
        if anterior<>nil
                then anterior^psig:=nuevo;
                else l:= nuevo;
End.
```

Insertar ordenado:

```
Procedure insertarOrdenado (var l:lista, elem:integer)
Var nuevo, actual, anterior: lista;
Begin
        new(nuevo);
        nuevo^dato:=elem;

        actual:=l;
        anterior:=nil;
        while actual<>nil and actual^dato<elem
                do begin
                        anterior:=actual;
                        actual:=actual^psig;
                        end;
        if anterior<>nil
                then begin
                        anterior^psig:=nuevo;
                        nuevo^psig:= actual;
                        end
                else begin
                        nuevo^psig:= l;
                        l:= nuevo;
                        end;
End.
```

Eliminar:

```
Procedure Eliminar:
begin
        {hacerlo.....}
end;
```

Guardar la lista en un archivo:

```
Procedure SalvarADisco(pi: lista; nom : cadena))
{ Guarda la lista en un archivo cuyo nombre es nom}
Var
        archi: file of integer;
        aux : lista;
begin
        assign(archi, nom);
        rewrite(archi);
        aux := pi;
        while aux <> nil
                do begin
                        write (archi, aux^.dato);
                        aux := aux ^.psig;
                end;
        close (archi);
end;
```

Armar la lista con datos de un archivo:

```
Procedure ArmoLista (var pi: lista; nom : cadena))
{ Arma una lista ordenada con los enteros almacenados en el archivo cuyo nombre es nom}
var
        archi: file of integer;
        x: integer;
begin
        assign(archi, nom);
        reset(archi);
        pi := nil;
        while not eof(archi)
                do begin
                        read (archi, x);
                        InsertarOrdenado(pi, x);
                end;
        close (archi);
end;
```

Ejercicio completo

Enunciado

Se tiene información sobre la asignación de cursos a profesores de un Instituto (nombre y apellido y cursos que dicta con su horario, a lo sumo tres). Todos los cursos se dictan los días lunes, miércoles y viernes; cada curso tiene el mismo horario todos los días. Por ejemplo, Arquitectura I, de 10 a 12 hs.

Se desea realizar un programa que permita modificar, para algunos profesores, uno de los cursos que dicta (el primero, el segundo o el tercero), siempre que el nuevo horario no se superponga con el de otro de los que dicta ese profesor). Se desea grabar en un archivo la información actualizada ordenada alfabéticamente.

Idea: Pensar en una solución general y de esta manera dividir el problema en módulos.

I)

Cargar la información en memoria, en una lista
Actualizar la información
Volver a grabar la información actualizada en disco

II)

/*Cargar la información en memoria, en una lista*/
abrir archivo de datos
while haya datos
    do begin
        leer datos
        guardarlo ordenado en la lista
    end

/*Actualizar la info*/
Ingresar pedido de modificación (nombre del profe, nro de curso, título y horario)
While haya pedidos
    do begin
        Analizar cambio en los cursos
        Ingresar otro pedido
    end

/*Volver a grabar la info actualizada en disco*/
(grabar la lista en el archivo)
.....

III)

/*Analizar cambio en los cursos*/

buscar el profesor en la lista
if esta
then begin
if no hay superposición horaria
then begin
modificar curso
guardarlo
end
else avisar que hay superposición horaria con el nuevo curso
end
else avisar que el profe no esta

IMPLEMENTACION EN PASCAL

```
program infoProfesores;

type
cadena = string [30];
curso = record
        titulo: string[30];
        horaI, horaF: real;
    end;

trescursos = array[1..3] of curso;

profesor = record
        nombre: cadena;
        cursos: trescursos;
    end;
archiProfes = file of profesor;
listaProfes =  ^ nodoProfe;
nodoProfe = record
        dato: profesor;
        psig: listaProfes;
        end;

var
    lista: listaProfes;

procedure CargarInfo (var lista: listaProfes);
var
    profe: profesor;
    archi: archiProfes;
begin
    Assign(archi, “profes.dat”);
    Reset(archi);
```

```
        lista = nil;
        while not eof(archi)
                do begin
                 read(archi, profe)
                 InsertarOrdenado(lista, profe) /*el que ya hicieron*/
                 end;
        Close( archi);
end;

procedure ActualizarInfo (var lista: listaProfes);
var
        profe: Cadena;
        nro: integer;
        cur: curso;

        begin
        writeln(" Ingrese el nombre del profesor a modificar (ZZ para finalizar)");
        read(profe);
        while profe <> "ZZ"
                do begin
                writeln(" Ingrese el nro del curso, título, hora inicial y hora final");
                read ( nro, cur.título, cur.horaI, cur.horaF);
                AnalizarCambio(lista, profe, nro, cur);
                writeln(" Ingrese el nombre del profesor a modificar (ZZ para finalizar)");
                read(profe)
                end;
        end;


procedure AnalizarCambio (var lista: listaProfes; profe: Cadena; n:integer;
nuevoCurso:curso);
var
        regProfe: profesor

        begin
        if  Buscar(lista, profe) /* busca por el nombre*/
                then begin
                        regProfe =Recuperar(lista, profe); /* recupera por el nombre*/
                        if not SuperposiciónHoraria (nuevoCurso, regProfe.cursos, n)
                        then begin
                                regProfe.cursos[n] = nuevoCurso
                                 InsertarOrdenado(lista, regProfe)
                                 end;
                                else writeln(" No se puede hacer el cambio por superposición
                                horaria");
                        end
                else writeln(" El profesor no está registrado");
        end;

function SuperposiciónHoraria (nuevo: curso, cursos : trescursos; n: integer): boolean;
```

```
        begin
{Se deberá testear que el horario del curso n no se superponga con los otros dos. }
        ..............
        end;

begin  /* del programa ppal */
        CargarInfo(lista);
        ActualizarInfo(lista);
        GrabarInfo(lista)
end.
```