UTN - FRRO

Algoritmos
y
Estructuras de Datos

* Registros
* Archivos

Prof. CLAUDIA DANIA
Magister en Docencia Universitaria
Analista Universitario de Sistemas
Licenciada en Sistemas de Información

OTRAS ESTRUCTURAS DE DATOS

REGISTROS

Aparte de la estructura de datos ARRAY, existe otra estructura similar porque también representan un grupo de elementos con un nombre en común que se denomina registros (RECORD); pero mientras que los elementos de un arreglo deben ser todos del mismo tipo, los elementos de un mismo registro pueden ser todos de diferentes tipos: simples y/o estructurados.
Por lo tanto, los registros son un tipo de datos estructurados, con un número fijo de componentes ó campos, no necesariamente del mismo tipo, a los que se accede por el nombre del campo y no por los subíndices.

```
TYPE nombre = RECORD          (* nombre es un identificador válido en Pascal *)
              identificador de campo : tipo;    (*nombre de 1 de las componentes del registro*)
              ....                              (*con su tipo asociado*)
              identificador de campo : tipo
              END;
```

```
Ejemplo:  TYPE persona = RECORD
                           nombre : ARRAY [1..20] OF Char;
                           legajo : Real;
                           cursa : Integer
                         END;
          VAR alumno : persona;
```

Para referirnos a un campo del registro, especificamos el nombre de la variable tipo registro y el nombre del campo separados entre sí, por un punto, denominando a esta expresión **Selector de Campo**.

Ejemplo: **alumno.nombre** es un selector y **alumno.cursa** es otro selector, los cuales referencian a dos campos distintos del mismo registro.

Dichos selectores pueden usarse en expresiones.

```
IF alumno.cursa > 2  THEN WRITE ('Correcto');
```

En el caso, donde el campo es un arreglo, se debe especificar el subíndice para acceder a cada carácter o elemento de dicho campo, dado que el mismo es estructurado.
Por ejemplo exhibir el campo nombre de 20 caracteres, del registro alumno:

```
FOR i := 1 TO 20 DO WRITE ( alumno.nombre[i] );
```

Debido a que los campos de un registro puden ser de cualquier tipo, se puede presentar el caso donde el tipo sea otro registro. A los registros que tienen este tipo de campos se los llama Registros Jerárquicos.

```
TYPE largo = ARRAY [1..20] OF Char;
     emple = RECORD
               nombre : largo;
               dire: RECORD
                       calle : largo;
                       nro : Integer;
                       piso : Integer
                     END;
               sueldo : Real
             END;
VAR empleado: emple;
```

Un selector de campo cualquiera sería: empleado.dire.piso, el cual me indicaría el piso de la dirección de un empleado.

ARREGLOS DE REGISTROS

Existe la posibilidad donde es ideal trabajar con arreglos y necesitamos definir a sus componentes como registros, por lo tanto el arreglo es de tipo estructurado (no de tipo simple).
Ejemplo: utilizando la definición anterior, del registro de un empleado, definimos la variable personal como un arreglo que almacenará los datos de 200 empleados.

```
TYPE largo = ARRAY [1..20] OF Char;

     emple = RECORD
                 nombre : largo;
                 dire: RECORD
                           calle : largo;
                           nro : Integer;
                           piso : Integer
                       END;
                 sueldo : Real
             END;

     empresa = ARRAY [1..200] OF emple;

VAR personal : empresa;
```

Siempre se trabaja con el nombre de la variable, que en caso de ser estructurada se indicará su posición, seguida de un punto, y a continuación el nombre del campo de trabajo.

```
personal [120].nombre[1]
(* especifica del empleado 120, el primer caracter de su nombre *)
```

ARCHIVOS

Es una estructura de datos, todos del mismo tipo, que puede guardarse en una unidad de almacenamiento masivo o memoria auxiliar (por ejemplo disco rígido, CD, DVD, pen, etc.).
De esta forma la información se preserva, pudiendo acceder a ella en cualquier momento y no se pierde al apagar el equipo. Esto es lo que da una verdadera funcionalidad a los programas ya que se puede conservar la información más allá del tiempo de corrida del programa e intercambiar dicha información con otros programas.
Además no es necesario definir previamente la longitud máxima de la estructura, dado que el límite de información a almacenar está dada por la capacidad física del medio de almacenamiento.

Existen dos tipos de archivos:
Permanentes o externos: Son los que se generan en un programa, se conservan en memoria auxiliar en forma permanente después de terminar la ejecución y pueden ser recuperados por el mismo programa que lo creó o por otro, en cualquier momento.
Deben transferirse a un programa pascal como parámetros, incluyendo el nombre del archivo en la cabecera del programa, separados por comas si hay más de uno y luego declararlos en la declarativa del programa.
Temporales o internos: Se guardan en la memoria principal del computador y se pierden cuando termina la ejecución del programa que lo creó.

Ambos tipos de archivos pueden ser organizados en **forma secuencial** o **de acceso directo.**
. Secuenciales: Todos los componentes se guardan uno después del otro. Para acceder a un elemento en particular se debe comenzar desde el primer elemento y recorrer el archivo hasta encontrar el dato buscado. Son los mas antiguos y su origen está vinculados con el soporte en cinta.
. Acceso Directo: Los elementos son recuperados directamente sin recorrer toda la estructura desde el principio. o sea que para acceder a un registro N cualquiera no se tiene que pasar por los N-1 registros anteriores. Son mucho más rápidos al momento de recuperar información.
Estos archivos tienen que tener sus registros de un tamaño fijo predeterminado de antemano.

ACTUALIZACION DE UN ARCHIVO SECUENCIAL
En Pascal un archivo secuencial puede ser utilizado como archivo de entrada o de salida pero nunca de ambos modos en el mismo programa, por lo tanto no se puede obtener un dato desde un archivo, modificarlo y grabarlo en el mismo archivo, ni tampoco leer un archivo hasta el final e intentar agregar más datos después del último leído.
Para ello se trabaja con dos archivos, uno que sería el **archivo viejo (archivo original o de entrada**), con información grabada el él, para leerla, modificarla si es necesario y un nuevo archivo en el cual se irán grabando los datos del archivo viejo, y/o datos nuevos que ingresen por teclado y se lo considerará **archivo de salida (archivo nuevo).**

La declaración de la estructura archivo, debe hacerse en la parte declarativa del programa pascal:

VAR nombre del archivo : FILE OF tipo;

ó

TYPE nombre de tipo : FILE OF tipo;
VAR nombre del archivo : nombre de tipo;

donde **nombre del archivo** debe ser un identificador válido y **tipo** es el tipo de datos de cada elemento individual del archivo, pueden ser de un tipo simple o estructurado, pero no pueden ser de tipo archivo.
Para poder acceder a cada dato dentro del archivo, tanto para leer como para guardar, se debe trabajar mediante un **buffe**r, que es quien transmite la información entre el archivo y el programa que lo invoca.
El buffer tiene el mismo nombre que el archivo y será del mismo tipo que los **componentes individuales** del archivo, solo se lo distingue por tener una flecha (h).

Operaciones Básicas con archivos

Declaración de un archivo: La declaración de un archivo directo consta de dos pasos:
1) Declaración del tipo adecuado: se realiza con las palabras reservadas **FILE O**F.
Esta declaración se realiza en la sección correspondiente a la declaración de tipos.

```
TYPE
    Enteros= FILE OF Integer;    archivo de enteros
                    ó
    Letra= ARRAY [1..100] of char;
    Caracters= FILE OF letra ;   archivo de array
```

2) Declaración de una variable de archivo de un tipo de archivo declarado.

```
VAR  Numeros: Enteros;
     Nombres: Caracters;
```

Normalmente no se desea crear archivos que puedan almacenar un solo tipo de datos ya que se requerirían varios archivos para poder registrar un conjunto de información. Para evitar este inconveniente se usa el tipo registro **(record**), que permiten grabar en un solo registro un grupo de datos de diferentes tipos.
Por ejemplo el registro de personas es:

```
TYPE  Datos = RECORD
          Nombre : String[40];
          Domicili : String[60];
          Edad : integer;
          EstCivil : char;
      END;
```

Y a continuación en el mismo Type, declarar un archivo del tipo Datos :

```
TYPE  Archiv = FILE OF Datos;
VAR   Personas : Archiv;
```

Asignación de un archivo
Declarada la variable archivo, se debe como primera actividad, vincular su identificador en el programa con un archivo físico. Esta operación asigna a un identificador de archivo un archivo real que es el que perdura en la memoria auxiliar.
El proceso de asignación establece una correspondencia entre la variable tipo archivo con un archivo externo.
Donde primero se declara **el nombre del archivo interno** por el que se conoce el archivo dentro del programa (el declarado) y segundo se declara **el nombre externo** con el que es reconocido el archivo por el sistema operativo, es una cadena que puede ser una constante, una variable o estar escrita directamente en el programa. Naturalmente debe cumplir con todas las reglas para nombrar un archivo. Debe cumplir con las reglas de los identificadores que se utilizan en el sistema operativo.
El nombre externo debe estar creado (debe existir) en la memoria auxiliar, ANTES de hacer la asociación utilizando el ASSIGN.

```
BEGIN
    ASSIGN (Personas, 'Personas.dat' );
```

En este ejemplo, se puede observar que el nombre interno (el que se usa dentro del programa) del archivo es **Persona**s, y el nombre externo **(el que se va a encontrar al explorar la memoria auxiliar**) es **Personas.dat.**
También es posible, indicar, al asociar el nombre interno con el nombre externo, una ubicación física determinada del archivo externo mediante la dirección completa:

```
ASSIGN(Personas, 'C:\Clientes\Personal.dat')
```

Indica que los datos del archivo Personas se almacenarán en el archivo de memoria auxiliar Personas.dat, dentro del subdirectorio Clientes de la unidad C.

Abrir archivos:
Una vez declarado y asignado un archivo ya es posible abrirlo.
En caso de querer abrir (usar) un archivo existente se utiliza el procedimiento **Rese**t .
En caso de querer abrir un archivo nuevo (se crea y se abre) se utiliza **Rewrite.**

Ambos procedimientos están predefinidos. Si al utilizar el procedimiento **Rewrite** el archivo asignado ya existía se borrará el anterior y se creará uno nuevo, por lo mismo se debe tener cuidado al momento de abrir estos archivos.
Las diferentes sintaxis son:

RESET (variabletipoarchivo);	se posiciona en el primer registro y no se borra.
REWRITE (variabletipoarchivo);	se posiciona en el primer registro y borra todo el contenido que estaba cargado.

Lectura y escritura de archivos.
Una vez que se ha abierto el archivo, para la lectura y escritura en un archivo de acceso directo se utilizan los procedimientos **Read, Write y** la función **Eo**f.

READ (variabletipoarchivo, variabletiporegistro);
WRITE (variabletipoarchivo, variabletiporegistro);
EOF (variabletipoarchivo);

La función **EO**F( ) es una función de tipo lógico que indica si el fin del archivo se ha alcanzado, devolviendo TRUE en caso afirmativo y FALSE en caso contrario. Generalmente esta función se utiliza cuando se avanza en un archivo registro por registro (recorrido secuencial), para determinar si ya se llegó al final del archivo.

Cerrar un archivo.
Todo archivo abierto debe ser cerrado cuando no se deja de usar:
CLOSE (variabletipoarchivo);

Mantenimiento de un archivo.
Con el paso del tiempo es necesario hacer operaciones que permitan mantener actualizados los datos que se guardan en un archivo directo, por lo tanto, es necesario modificar o actualizar datos después que el archivo ha sido creado (modificar uno o más campos, borrar un registro, insertar un nuevo registro, o visualizar los valores de los campos de un registro). Para poder ejecutar las acciones detalladas es necesario utilizar una serie de procedimientos y funciones que se detallan a continuación.

Tamaño de un archivo
La función **FileSize** regresa el tamaño actual de un archivo, es decir devuelve el número de registros contenidos en éste. Al momento de abrir un archivo nuevo la función **FileSize** regresa el valor 0, lo que significa que el archivo no tiene datos guardados en él.

tamaño := FILESIZE (variabletipoarchivo);

Posicionamiento en el interior de un archivo.
La función **FilePos** devuelve el número del registro actual del archivo.

posición := FILEPOS(variabletipoarchivo);

Para moverse (posicionarse) a un registro determinado se utiliza la función **Seek.**

SEEK (variabletipoarchivo, númeroderegistro); **seek** (personas,17);

Sobre la base que cada registro de un archivo esta referenciado por un número específico comenzando desde el registro 0 y aumentando de 1 en 1.
Para moverse al final del archivo para agregar un nuevo registro se utiliza este mismo comando con el parámetro *númeroderegistro* como sigue:

SEEK (Variable tipoArchivo, **FILESIZ**E(VariabletipoArchivo));

Ejemplo 1: Se define la estructura de un archivo que va a guardar los datos de un grupo de alumnos y se permite la carga de dichos datos.

```
Program eje1(input, output);
type
        alumnos = RECORD
                legajo: integer;
                nombre: string[30];
                edad: integer;
        end;
var
        alumno: alumnos;
        archivo: file of alumnos;
        ba: char;
BEGIN
        assign(archivo, 'd:\pascal\alumnos.dat');
        rewrite(archivo);
        ba:='s';
        While ba='s' do
        BEGIN
                write('Ingrese el legajo del alumno : ');readln(alumno.legajo);
                write('Ingrese el nombre del alumno : ');readln(alumno.nombre);
                writeln('Ingrese la edad del alumno : ');readln(alumno.edad);
                write(archivo, alumno);
                Write ('¨Ingresa datos de otro alumno? ( s=SI n=NO)');
                repeat
                        read (ba)
                until (ba = 's') or (ba = 'n');
        END;
        close(archivo);
        writeln('NUEVOS REGISTROS INSERTADOS´);
        readln;
END.
```

Ejemplo 2: Ahora se pretende presentar la forma en que se realiza una búsqueda de un alumno, en el archivo cargado previamente (en el ejemplo 1), utilizando como dato de búsqueda la posición del registro donde fueron guardados sus datos.

```
Program eje2 (input, output);
Type
        alumnos = RECORD
                legajo:integer;
                nombre:string[30];
                edad: integer;
        end;
var
        archivo: file of alumnos;
        alumno:alumnos;
        pos:integer;
        ba: char;
BEGIN
        ba:= 's';
        assign(archivo,'d:\pascal\alumnos.dat');
        reset(archivo);
        While ba='s' do
        BEGIN
                write('nro de registro a buscar');
                readln(pos);
                seek(archivo,pos-1);
                read(archivo,alumno);
                write('Legajo : ');writeln(alumno.legajo);
                write('Nombre : ');writeln(alumno.nombre);
                write('Edad : ');writeln(alumno.edad);
```

```
            write (‘¨Desea datos de otro alumno? ( s=SI n=NO)’);
            repeat
                  read (ba)
            until (ba = ‘s’) or (ba = ‘n’);
      END;
      close(archivo);
ENd.
```

Ejemplo 3: Se creará un archivo que contenga números enteros y luego los muestre, con la particularidad que, en la muestra solo aparecerán los números pares.

```
Program eje3 (input,output);
Type
      Arc= File of integer;
var
      Archivo: Arc;
      Numero : Integer;
      Fin:char;
Procedure Alta_num;
Var
      Rta : Char;
Begin
      Rewrite(Archivo);
      Repeat
            write(‘Introduzca un número entero : ‘);readln(Numero);
            write(Archivo, Numero);
            writeln(‘¨ Desea guardar otro número (S/N) : ‘);
            repeat
                  read (rta)
            until (rta = ‘S’) or (rta =’N’);
      until rta=’N’;
End;
Procedure Mues_num
Begin
      Writeln(‘Números enteros y pares del archivo’);
      Seek(Archivo,0);
      while not EOF(Archivo) do
            begin
                  Read(Archivo, Numero);
                  if Numero Mod 2 = 0 then writeln(Numero:6);
            end;
End;
BEGIN
      Assign(Archivo,’d:\pascal\Numeros.dat’);
      Alta_num;
      Mues_num;
      write (‘FIN DEL PROGRAMA’);
      READLN (fin);
      Close(Archivo)
END.
```

Operaciones para crear un archivo externo

Existen al menos dos formas de crear un archivo externo para luego utilizar el Assign y asociar una variable declarada tipo archivo con dicho archivo externo.

1) **Desde el Pascal, como parte del programa. Este fragmento de Programa puede ser agregado al Ejemplo 1, para crear el archivo antes de insertar datos.**
Se utilizará la directiva del compilador **{$I}** .Esta directiva cuando esta activa (por defecto) verifica los errores de entrada y salida (querer abrir un archivo que no exista es un error de entrada y salida).
Entonces lo que hacemos es, desactivar el chequeo de error **{$I-}** e intentar abrir el archivo, si hay error es que no se pudo abrir porque no se encontró, entonces se crea.
Para saber si hubo un error se comprueba el valor de la function **IORESULT** que devuelve un entero que representa el estado de la ultima operación de entrada y salida, si dicho valor es diferente de 0 significa que hubo un error. El numero de error cuando no se encuentra un archivo es el 2.
Por último se debe volver a activar la directiva **{$I+}**

```
Begin
        assign(archivo,'alumnos.dat');
        {$I-}
        reset(archivo);
        If ioresult =2 then rewrite(archivo);
        {$I+}
end;
```

Esto debe ser expresado como un procedimiento e invocado al inicio del programa.

2) Una forma fácil de crear un archivo externo es generar un archivo vacío. Para ello, ejecute el Pascal y desde la barra de Menú seleccione la opción SAVE:
A continuación escriba el nombre que desee para el archivo externo, archivo que va a contener los datos ingresados.

Otras operaciones de gestión de archivos

Pascal tiene operaciones que permiten manipular archivos y directorios en disco similar al sistema operativo MS-DOS (aunque el ambiente es WINDOWS).

Se pueden realizar las siguientes operaciones con archivos y directorios:
o **Erase**(nomvararchivo) Borra un archivo
o **Rename**(nomvararchivo,'nombrenuevoarchivo') Renombra
o **Chdir** (directorio) Cambia Directorio
o **Mkdir** (directorio) Crea un Directorio
o **Rmdir** (directorio) Borra un Directorio

o **SENTENCIA WITH**
Cuando se trabaja con variables tipo registro se tiene que acceder varias veces a uno o más campos de dicha variable en una pequeña sección de código; y como los selectores de campo suelen ser muy largos, la sentencia WITH nos permite abreviar la notación, especificando una vez el nombre del registro y usando luego los identificadores de campo. La forma es la siguiente:

```
WITH variable registro DO sentencia;

WITH empleado DO
      FOR i := 1 TO 20 DO
                READ (sueldo);                    (* no leo de la forma empleado.sueldo *)
```

La sentencia WITH puede anidarse. Por ejemplo

```
WITH variable registro1 DO
      WITH variable registro2 DO
            sentencia;

            ó
WITH (variable registro1, variable registro2) DO sentencia:
```

o **BUCLES EOF**

EOF es una función booleana del Pascal que verifica si se ha leido el último dato. Se hace TRUE cuando el siguiente carácter leido es el carácter especial fin de archivo <eof>, que es el último en un fichero de datos. Este carácter lo coloca el sistema.
Esta función se usa para controlar los bucles de lectura donde no sabemos de antemano cuantos datos hay.
Veamos un ejemplo con READLN (se usa porque sabemos como están organizados los datos en líneas).

```
WHILE NOT EOF DO          (* El NOT es porque EOF termina por verdadero *)
    BEGIN
        READLN (valor1, valor2, ..);
        .....
    END;

ó

REPEAT
    READLN (variabletipoarchivo, variabletiporegistro);
    .....
    .....
UNTIL EOF(variabletipoarchivo) (* ejemplo de recorrido secuencial de un archivo hasta su final*)
```

En el caso de no saber como están los datos y además son numéricos, se usa la técnica de lectura adelantada para detectar el EOF y no procesarlo.

```
READ (valor);
WHILE NOT EOF DO
    BEGIN
        .....
        READ (valor)
    END;
```

o **BUCLES EOLN**

EOLN es una función booleana del Pascal que verifica si se ha leido el último carácter de una línea, se hace TRUE si el siguiente carácter a leer es el <eoln>. Se usa para controlar un bucle en el que se procesan caracteres, y como se hace carácter a carácter no es necesario la lectura adelantada.
En caso de leer el <eoln>, se convierte en blanco antes de que se almacene.

```
WHILE NOT EOLN DO          (* el NOT es porque EOLN finaliza por verdadero *)
    BEGIN
        READ (ch);
        write (CH)
    END;
```