

SEP

SEIT

DGIT

INSTITUTO TECNOLÓGICO DE NUEVO  
LAREDO

DEPTO. DE SISTEMAS Y COMPUTACIÓN



## **“Manejo de Archivos en Pascal”**

Por:

***Ing. Bruno López Takeyas, M.C.***

<http://www.itnuevolaredo.edu.mx/takeyas>

Email: [takeyas@itnuevolaredo.edu.mx](mailto:takeyas@itnuevolaredo.edu.mx)

## TABLA DE CONTENIDO

	<b>Pág.</b>
<b>Tabla de figuras.....</b>	<b>5</b>
<b>Prefacio.....</b>	<b>7</b>
 <b>1.- CONCEPTOS BÁSICOS DE ARCHIVOS.....</b>	 <b>8</b>
1.1. ¿Cómo surge la necesidad de utilizar archivos?.....	8
1.2. Relación entre la memoria principal, el microprocesador y dispositivos de almacenamiento secundario.....	9
1.3. Definiciones de datos, registros y archivos.....	10
1.4. Analogías de archivos y archiveros.....	12
1.5. Apertura de archivos.....	16
1.6. Clasificación de archivos por tipo de contenido.....	17
1.6.1. Archivos de texto.....	18
1.6.2. Archivos binarios.....	18
1.7. Clasificación de archivos por tipos de acceso.....	19
1.7.1. Archivos secuenciales.....	20
1.7.1.1. Consulta o recorrido secuencial.....	20
1.7.2. Archivos directos (relativos, de acceso directo o aleatorios)...	22
1.7.2.1. Direcciones lógicas y direcciones físicas.....	22
1.7.2.2. Control de direcciones lógicas.....	23
1.7.2.3. Consulta directa.....	24
 <b>2.- FUNCIONES DE MANEJO DE ARCHIVOS EN PASCAL.....</b>	 <b>26</b>
2.1. Declaración del alias del archivo.....	26
2.2. Funciones de manejo de archivos en Pascal.....	26
2.2.1. La función <code>Assign</code> y modos de apertura de archivos.....	27
2.2.2. Validar la apertura de un archivo.....	28
2.2.3. Cierre de archivos usando <code>Close</code> .....	29
2.2.4. Escritura de registros usando <code>Write</code> .....	29
2.2.4.1. Vaciando los buffers con <code>Flush</code> .....	31
2.2.5. Lectura de registros usando <code>Read</code> .....	31
2.2.6. Reposicionando el apuntador mediante <code>Seek</code> .....	31
2.2.6.1. Conociendo la posición del apuntador del archivo con la función <code>FilePos</code> .....	34
2.2.7. Detectando el final del archivo con <code>Eof</code> .....	35
2.2.8. Cambiando nombres de archivos mediante <code>Rename</code> .....	35
2.2.9. Eliminando archivos con la función <code>Erase</code> .....	36
 <b>3.- APLICACIONES DE ARCHIVOS EN PASCAL.....</b>	 <b>38</b>

3.1. Declaraciones globales.....	38
3.2. Archivos secuenciales en Pascal.....	39
3.2.1. ALTAS secuenciales.....	40
3.2.1.1. Diagrama de flujo de la rutina de ALTAS secuenciales.....	40
3.2.1.2. Codificación de la rutina de ALTAS secuenciales.....	42
3.2.2. CONSULTAS secuenciales.....	43
3.2.2.1. Diagrama de flujo de la rutina de CONSULTAS secuenciales.....	43
3.2.2.2. Codificación de la rutina de CONSULTAS secuenciales.....	43
3.2.3. LISTADO secuencial.....	44
3.2.3.1. Diagrama de flujo de la rutina de LISTADO secuencial.....	44
3.2.3.2. Codificación de la rutina de LISTADO secuencial.....	46
3.2.4. MODIFICACIONES de datos en un archivo secuencial.....	47
3.2.4.1. Diagrama de flujo de la rutina de MODIFICACION secuencial.....	47
3.2.4.2. Codificación de la rutina de MODIFICACIÓN secuencial.....	49
3.2.5. BAJAS de registros en un archivo secuencial (bajas ógicas y bajas físicas).....	50
3.2.5.1. Diagrama de flujo de la rutina de BAJAS lógicas en un archivo secuencial.....	51
3.2.5.2. Codificación de la rutina de BAJAS lógicas en un archivo secuencial.....	53
3.2.5.3. Diagrama de flujo de la rutina de BAJAS físicas en un archivo secuencial (compactar).....	54
3.2.5.4. Codificación de la rutina de BAJAS físicas en un archivo secuencial (compactar).....	56
3.3. Archivos directos en Pascal.....	57
3.3.1. ALTAS directas.....	57
3.3.1.1. Diagrama de flujo de la rutina de ALTAS directas.....	57
3.3.1.2. Codificación de la rutina de ALTAS directas.....	59
3.3.2. CONSULTAS directas.....	60
3.3.2.1. Diagrama de flujo de la rutina de CONSULTAS directas.....	60
3.3.2.2. Codificación de la rutina de CONSULTAS directas....	62
3.3.3. MODIFICACIONES directas.....	63
3.3.3.1. Diagrama de flujo de la rutina de MODIFICACIONES directas.....	63
3.3.3.2. Codificación de la rutina de MODIFICACIONES directas.....	65
3.3.4. BAJAS de registros en un archivo de acceso directo (bajas lógicas).....	66
3.3.4.1. Diagrama de flujo de la rutina de BAJAS lógicas directas.....	67

3.3.4.2. Codificación de la rutina de BAJAS lógicas directas..	69
<b>4.- CONCLUSIONES.....</b>	<b>71</b>
<b>5.-BIBLIOGRAFÍA.....</b>	<b>72</b>

## TABLA DE FIGURAS

<b>No.</b>	<b>Descripción</b>	
1	Interacción entre la memoria, microprocesador y archivos.....	8
2	Formato del registro de Productos.....	10
3	Declaración del registro de Productos.....	10
4	Declaración del registro de Productos.....	11
5	Cuadro comparativo de archivos y archiveros.....	12
6	Apertura de archivos.....	16
7	Clasificación de archivos por contenido.....	17
8	Clasificación de archivos por tipo de acceso.....	19
9	Diagrama de flujo de rutina de consulta secuencial.....	20
10	Ejemplo de cálculo del espacio ocupado por un registro.....	21
11	El lenguaje Pascal maneja archivos con direcciones lógicas.....	22
12	Direcciones lógicas y físicas de un archivo.....	23
13	Cálculo de la dirección física a partir de la dirección lógica.....	23
14	Diagrama de flujo de rutina de consulta directa.....	24
15	Las funciones Assign y Reset.....	26
16	Validar la apertura de un archivo.....	27
17	La función Write.....	28
18	La función SEC.....	30
19	La función FilePos.....	31
20	La función Rename.....	32
21	La función Erase.....	32
22	Declaraciones globales de las aplicaciones.....	34
23	Diagrama de flujo de rutina de alta secuencial.....	36
24	Codificación de la rutina de altas secuenciales.....	37
25	Codificación de la rutina de consultas secuenciales.....	38
26	Diagrama de flujo de rutina de listado secuencial.....	40
27	Codificación de la rutina de listado secuencial.....	41
28	Diagrama de flujo de rutina de modificación secuencial.....	43
29	Codificación de rutina de modificación secuencial.....	44
30	Diagrama de flujo de rutina de baja lógica secuencial.....	47
31	Codificación de rutina de baja lógica secuencial.....	48
32	Diagrama de flujo de rutina de baja física secuencial (compactar)....	51
33	Codificación de rutina de baja física secuencial (compactar).....	52
34	Inserción de registros en blanco desde el final del archivo.....	54
35	Diagrama de flujo de rutina de altas directas.....	56
36	Codificación de rutina de altas directas.....	57
37	Diagrama de flujo de rutina de consultas directas.....	59
38	Codificación de rutina de consultas directas.....	60
39	Diagrama de flujo de rutina de modificación directa.....	62
40	Codificación de rutina de modificaciones directas.....	63
41	Diagrama de flujo de rutina de baja lógica directa.....	66
42	Codificación de rutina de baja lógica directa.....	67

## PREFACIO

Durante el tiempo que he impartido la materia de “Administración de Archivos” en la carrera de Ingeniería en Sistemas Computacionales (ISC) en el Instituto Tecnológico de Nuevo Laredo (ITNL), me he percatado de las deficiencias de los alumnos para programar archivos y, aunque es necesario dominar este aspecto de programación para aplicarlo en la materia, no es limitante o requisito estricto para cursarla, ya que la retícula así lo señala. Además estoy enterado que los temas de archivos pertenecen a la última unidad programática de las materias previas de Programación I y II y que debido a lo extenso de esos programas de estudio, no se comprenden completamente los temas relacionados con archivos. Debido a lo anterior, presento este documento basado en un cúmulo de experiencias y dudas planteadas por alumnos que tiene como finalidad reforzar los conocimientos de programación de archivos en Pascal para aplicarlos a necesidades específicas de la materia “Administración de Archivos”.

*Ing. Bruno López Takeyas, M.C.*

*<http://www.itnuevolaredo.edu.mx/takeyas>*

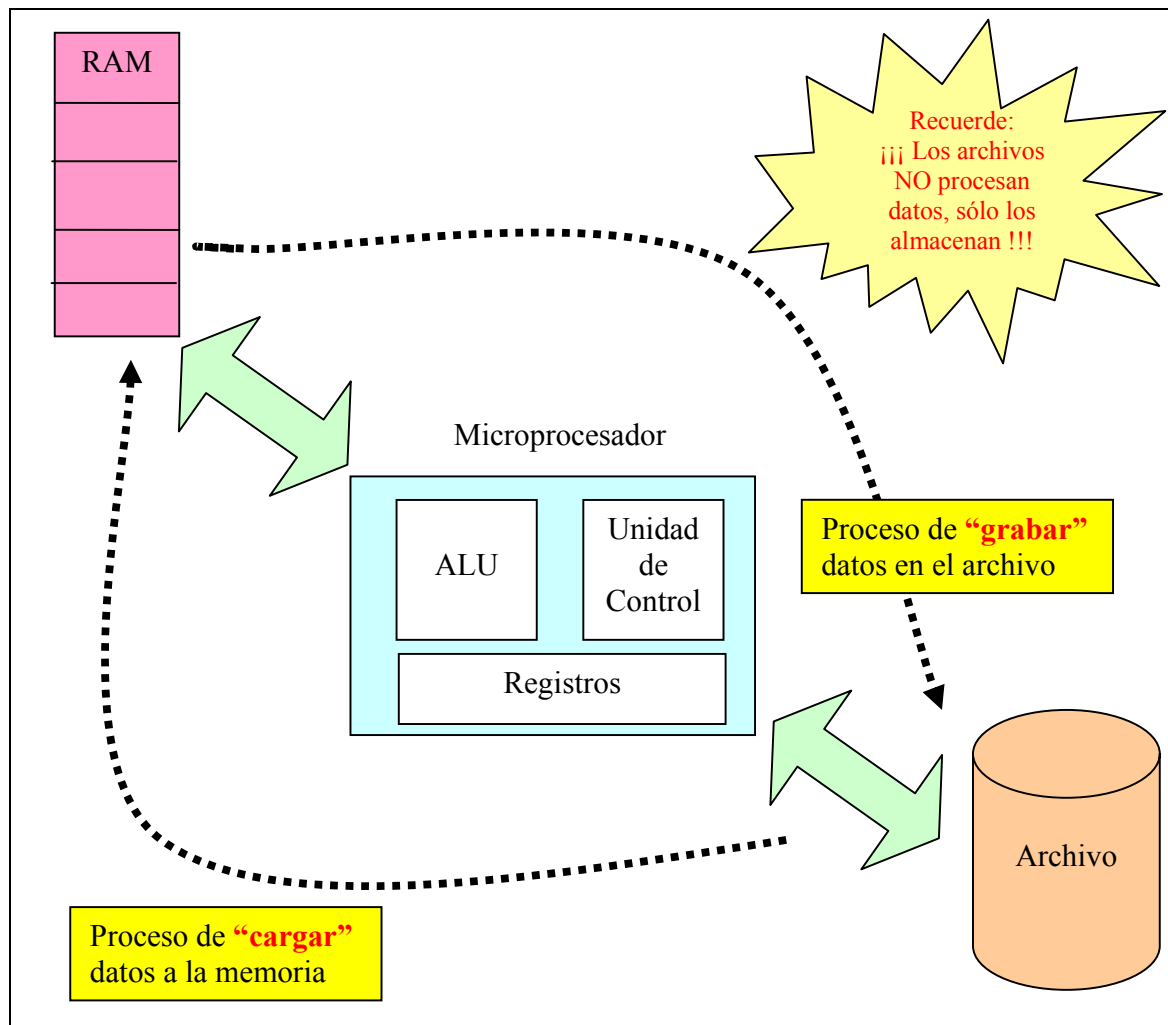
*[takeyas@itnuevolaredo.edu.mx](mailto:takeyas@itnuevolaredo.edu.mx)*

# **1.CONCEPTOS BÁSICOS DE ARCHIVOS**

Esta sección presenta las generalidades relacionadas con archivos antes de empezar a utilizarlos y programarlos. Es necesario involucrarse con la terminología relacionada como archivo, registro, campo, etc. También es recomendable conocer las clasificaciones generales y las operaciones fundamentales con archivos.

## **1.1. ¿Cómo surge la necesidad de utilizar archivos?**

Hasta antes de la materia de Administración de Archivos, muchas de las aplicaciones que los alumnos de ISC han programado han sido usando la memoria principal o memoria RAM como único medio de almacenamiento (usando variables, arreglos o estructuras de datos mas complejas), con el inconveniente que esto representa: la volatilidad de la memoria RAM; es decir, si se apaga la computadora se pierden los datos. Además, algunas aplicaciones exigen transportar los datos de una computadora a otra. De ahí surge la necesidad de almacenar dichos datos de forma permanente que permita retenerlos en ciertos dispositivos de almacenamiento secundario por un período de tiempo largo sin necesidad de suministrarles energía, de tal forma que permitan transportarlos y utilizarlos en otro equipo computacional.



*Fig. 1. Interacción entre la memoria principal, el microprocesador y los archivos*

## 1.2. Relación entre la memoria principal, el microprocesador y dispositivos de almacenamiento secundario

Existe una estrecha relación entre la memoria principal, el microprocesador y los dispositivos de almacenamiento secundario ya que el procesamiento que realiza una computadora es tarea absoluta del microprocesador en conjunción con



la memoria principal; es decir, los dispositivos de almacenamiento secundario (diskettes, discos duros, CDs, flash drives, etc.) **no procesan datos, sólo los almacenan.** En estos dispositivos sólo se reflejan los datos previamente procesados y funcionan exclusivamente como una bodega. Esto repercute de manera significativa al momento de programar archivos, ya que para hacerle modificaciones a los datos de un registro previamente almacenado es necesario primero **“cargarlo”** en la memoria principal, es decir, localizar el registro en el archivo y leerlo para colocar sus datos en la memoria RAM, ahí modificarlo y posteriormente **grabarlo en la misma posición** en la que se encontraba, sin embargo estas operaciones no se realizan directamente, sino a través de la unidad aritmética-lógica, la unidad de control y los registros del microprocesador (Fig. 1).

### 1.3. Definiciones de datos, registros y archivos

- **Datos:** Básicamente se refieren a los testimonios individuales relacionados con hechos, ya sean características de ciertos objetos de estudio o condiciones particulares de situaciones dadas. Los elementos individuales de los archivos se llaman datos o campos. Por ejemplo un cheque de un banco tiene los siguientes campos: Cuenta habiente, Número de cheque, Fecha, Persona a la que se le paga, Monto numérico, Monto con letra, Nota, Identificación del banco, Número de cuenta y Firma. Cada campo es definido por un tipo de dato.
- **Registro:** Es el conjunto completo de datos relacionados pertenecientes a una entrada. P. ejem. Un almacén puede retener los datos de sus productos en registros de acuerdo al formato mostrado en la Fig. 2.

No_prod	Descrip	Cantidad	Precio	Garantia
Entero	Cadena [30]	Entero	Real	Caracter

*Fig. 2. Formato del registro de Productos*

El registro de la Fig. 2 puede ser declarado globalmente en Pascal utilizando Record (Fig. 3).

```

Type Tipo_registro = Record
  no_prod : integer;
  descrip : string[30];
  cantidad: integer;
  precio  : real;
  garantia: char;
End;

Var Registro : Tipo_registro;

```

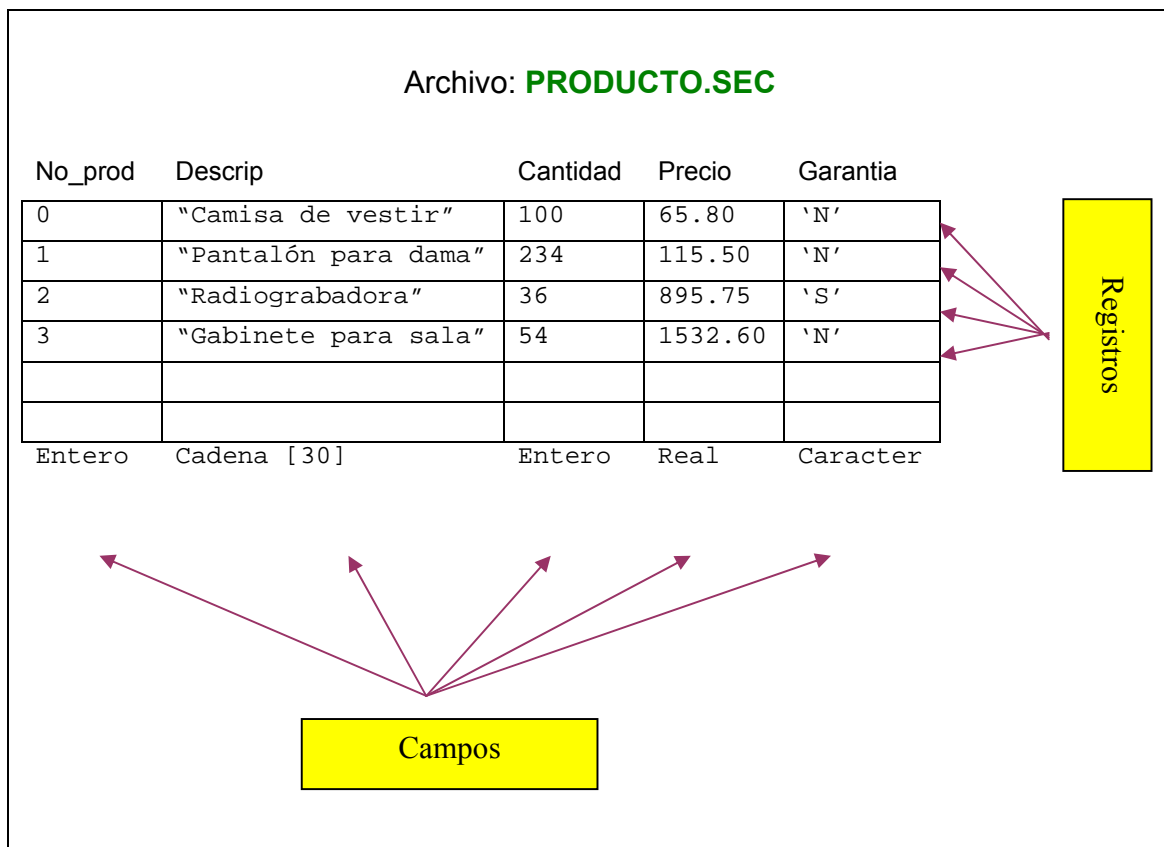
Declaración del tipo de dato (en este caso del tipo de registro)

Declaración de la variable "Registro" de tipo "Tipo\_registro"

*Fig. 3. Declaración del registro de Productos*

- **Archivo:** En términos computacionales es una colección de datos que tiene un nombre y se guardan en dispositivos de almacenamiento secundario, los cuales pueden ser magnéticos, ópticos, electrónicos, etc. P. ejem. Diskettes, discos duros, CD's, ZIP drives, flash drives, memory sticks, etc.

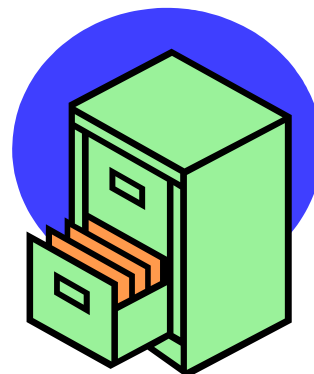
P. ejem. La Fig. 4 muestra la estructura de un archivo con registros homogéneos de Productos (con la misma estructura) declarados previamente (Fig. 3).







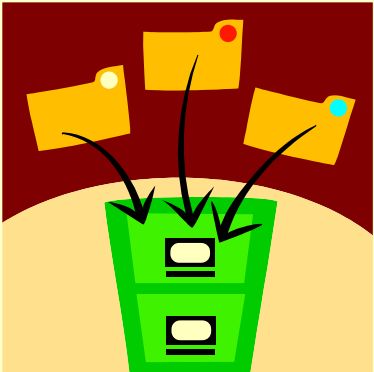
*Fig. 4. Formato del registro de Productos*

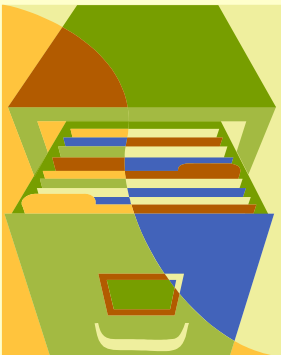
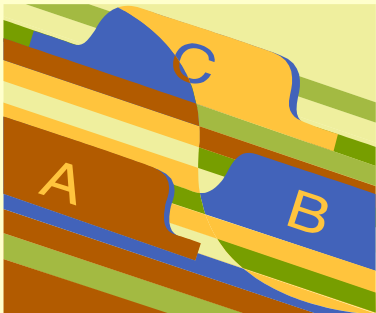
## 1.4. Analogías de archivos y archiveros

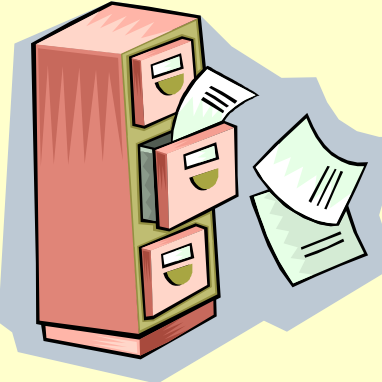

El modo de operación de un archivo puede ser asociado con el de un archivero en una oficina, ya que ambos almacenan datos y operan de forma semejante. De tal forma que muestran las siguientes operaciones, acciones similares y comparaciones:



<b>Operación o acción</b>	<b>Archivero</b>	<b>Archivo computacional</b>
<p><b>Identificar la localización de la información</b></p> 	 <p>Localizando el archivero en particular que contiene las carpetas con la información que se solicita, ya que una oficina puede tener varios archiveros debidamente clasificados e identificados</p>	 <p>Identificando la base de datos correspondiente a la información que se solicita. Una base de datos es una colección de archivos relacionados. P. Ejem. Profesores, alumnos y materias están correlacionados.</p>
<p><b>Identificar el lugar exacto donde se encuentra la información</b></p>	<p>Regularmente un archivero contiene varios cajones, cada uno con información debidamente clasificada y ordenada.</p>	<p>Se recomienda que los archivos contengan datos relacionados con un objeto de interés en particular y no de varios. P. Ejem. Sólo datos de ALUMNOS.</p>

<p><b>Nombres</b></p> 	<p>Se pueden colocar etiquetas a los cajones de los archiveros para identificar su contenido de tal forma que indique el contenido o la clasificación de las carpetas que contiene.</p>	<p>Los nombres de los archivos están regidos por el sistema operativo, pero regularmente se componen del nombre principal y su extensión, la cual puede ser de cualquier tipo, es decir, el usuario puede colocarle la extensión que desee ya sea DAT, TXT, BIN, JK8, etc. Sin embargo se recomienda que tanto el nombre como la extensión sean relevantes al contenido del archivo. P. Ejem. ALUMNOS.DAT, ARCHIVO.TXT</p>
<p><b>Operaciones</b></p> 	<p>En un archivero se pueden agregar, extraer o cambiar documentos de las carpetas.</p>	<p>Básicamente un archivo solo tiene 2 operaciones:</p> <ul style="list-style-type: none"> <li>• Lectura</li> <li>• Escritura</li> </ul> <p>Las demás operaciones se realizan como consecuencia de éstas.</p>

<p><b>Apertura</b></p> 	<p>Obviamente cuando se requiere agregar o consultar carpetas del cajón de un archivero, es necesario primero abrirlo.</p>	<p>Para acceder los datos de un archivo es necesario abrirlo. Existen varios modos de apertura de los archivos dependiendo de las operaciones que se deseen realizar en él.</p>
<p><b>Clasificación de los datos</b></p> 	<p>Los cajones de los archiveros tienen separadores o pequeñas pestañas para identificar las carpetas. Estas facilitan el acceso, ya sea la inserción o la extracción de un carpeta en particular.</p>	<p>Los datos pueden ser almacenados de muchas formas diferentes en los archivos y de esto depende la facilidad (o dificultad) que el archivo muestre para ciertas operaciones de acceso. A estas formas de almacenamiento se les conoce como "organización del archivo".</p>

<p><b>Cierre</b></p> 	<p>Cuando ya no se desea utilizar un cajón de un archivero es necesario cerrarlo, ya que de no hacerlo, se corre el riesgo de dañar o perder la información.</p>	<p>Cuando se termina de utilizar un archivo es necesario cerrarlo. De esa forma se vacía la memoria caché y se asegura almacenar y proteger los datos.</p>
<p><b>Seguridad</b></p> 	<p>Algunos archiveros cuentan con un candado que permite asegurar los cajones de aperturas no deseadas por personal no autorizado.</p>	<p>Dependiendo del sistema operativo, se pueden aplicar restricciones de seguridad y acceso a los archivos dependiendo de los usuarios; es decir, se establecen políticas de acceso a usuarios.</p>

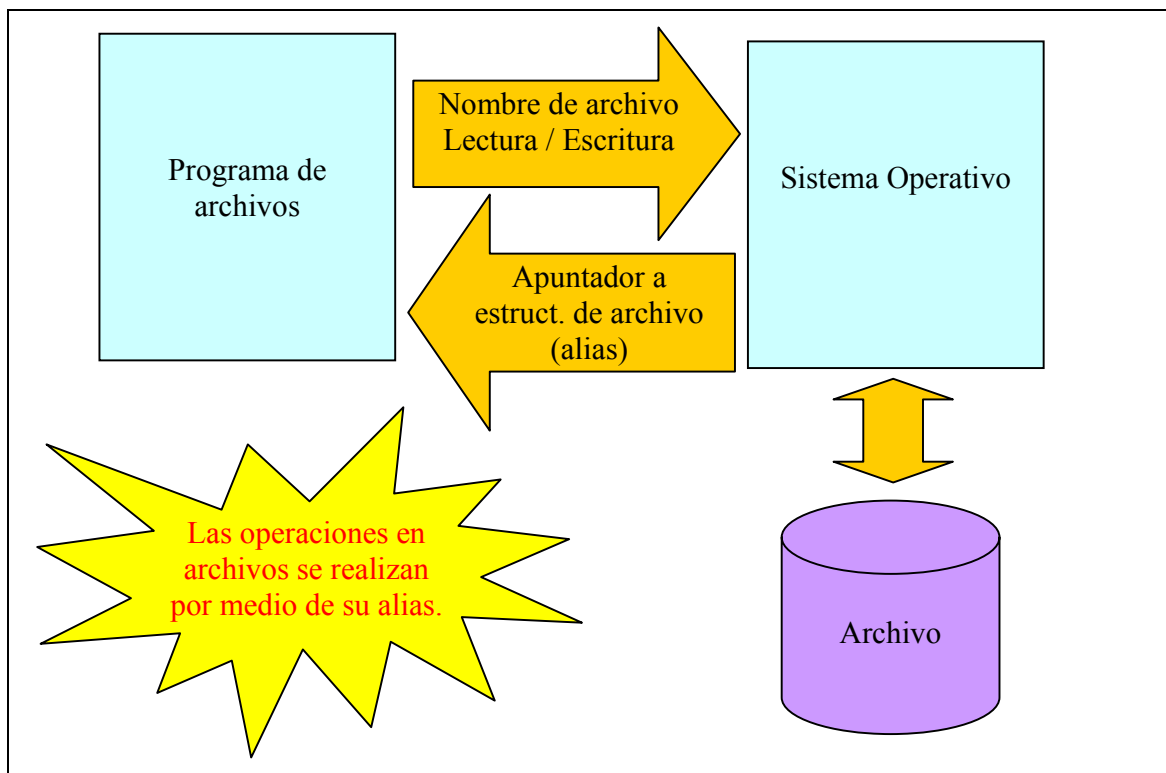
*Fig. 5. Cuadro comparativo de archivos y archiveros*

## 1.5. Apertura de archivos

Antes de escribir o leer datos de un archivo es necesario abrirlo. Al abrir el archivo se establece comunicación entre el programa y el sistema operativo a cerca de cómo accesarlo. Es necesario que el programa le proporcione al sistema operativo el nombre completo del archivo y la intención de uso (leer o escribir datos), entonces se definen áreas de comunicación entre ellos. Una de estas

áreas es una estructura que controla el archivo (**alias del archivo**), de esta forma cuando se solicita una operación del archivo, se recibe una respuesta que informa el resultado mediante un apuntador. Cada archivo abierto requiere un alias para poder realizar operaciones en él (Fig.6).

La estructura del archivo contiene información del archivo que se está usando, su tamaño actual y la localización de los buffers de datos.

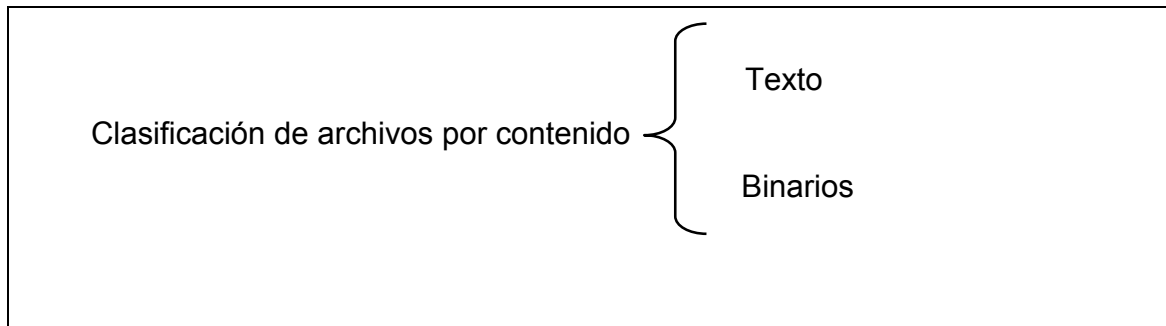


*Fig. 6. Apertura de archivos*

## 1.6. Clasificación de archivos por tipo de contenido

Existen muchas clasificaciones de archivos de acuerdo a diferentes criterios o aplicaciones. Aquí se presenta una muy importante: de acuerdo al contenido.





*Fig. 7. Clasificación de archivos por contenido*

### 1.6.1. Archivos de texto

Son aquellos que pueden contener cualquier clase de datos y de tal manera que son “entendibles” por la gente. Los datos en un archivo de texto se almacenan usando el código ASCII, en el cual cada carácter es representado por un simple byte. Debido a que los archivos de texto utilizan el código ASCII, se pueden desplegar o imprimir. En este tipo de archivos, todos sus datos se almacenan como cadenas de caracteres, es decir, los números se almacenan con su representación ASCII y no su representación numérica, por lo tanto no se pueden realizar operaciones matemáticas directamente con ellos. P. ejem. Si se guarda el dato 3.141592 en un archivo de texto, se almacena como “3.141592” y nótese que

...

**3.141592 ≠ “3.141592”**

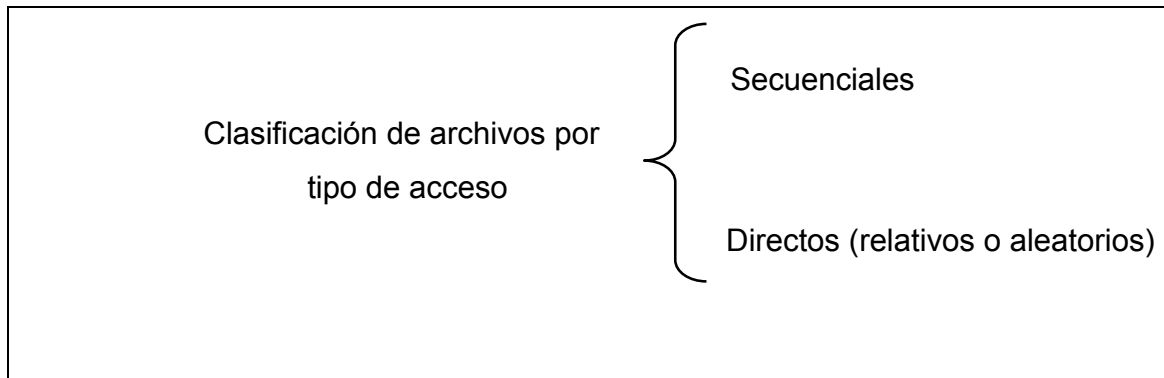
### 1.6.2. Archivos binarios

Este tipo de archivos almacenan los datos numéricos con su representación binaria. Pueden ser archivos que contienen instrucciones en lenguaje máquina

listas para ser ejecutadas. Por ejemplo, cuando escribimos un programa en un lenguaje en particular (como C++, Pascal, Fortran, etc), tenemos las instrucciones almacenadas en un archivo de texto llamado programa fuente, pero una vez que lo sometemos a un proceso de compilación y ejecución nuestro programa lo trasladamos a un programa ejecutable (en lenguaje máquina), que es directamente entendido por la computadora y se crea un archivo binario. En este tipo de archivos también se pueden almacenar diferentes tipos de datos incluyendo datos numéricos; sin embargo, cabe destacar que los datos numéricos se graban con su representación binaria (no con su representación ASCII), por tal razón, cuando se despliegan con un editor de textos o por medio de comandos del sistema operativo, aparecen caracteres raros que no se interpretan. P. ejem. Si se guarda el dato 27 en un archivo binario, se almacena como 00001111 y no como "27".

## **1.7. Clasificación de archivos por tipo de acceso**

De acuerdo a la forma de acceder los datos de los archivos, éstos se clasifican en secuenciales o directos (también conocidos como de acceso directo, relativos o aleatorios). En esta sección no se pretende analizar las diferentes estructuras de datos involucradas en estas clasificaciones de archivos ni desarrollar aplicaciones complejas debidamente diseñadas usándolos, sino conocer esencialmente sus conceptos teóricos y la forma de manejarlos.



*Fig. 8. Clasificación de archivos por tipo de acceso*

### 1.7.1. Archivos secuenciales

Como su nombre lo indica, en este tipo de archivos los registros se graban en secuencia o consecutivamente y deben accesarse de ese mismo modo, es decir, conforme se van insertando nuevos registros, éstos se almacenan al final del último registro almacenado; por lo tanto, cuando se desea consultar un registro almacenado es necesario recorrer completamente el archivo leyendo cada registro y comparándolo con el que se busca. En este tipo de archivo se utiliza una marca invisible que el sistema operativo coloca al final de los archivos: **EOF** (End of File), la cual sirve para identificar dónde termina el archivo

#### 1.7.1.1. Consulta o recorrido secuencial

A continuación se muestra un diagrama de flujo de una rutina de consulta de un registro en particular mediante un recorrido secuencial. En esta rutina se maneja un archivo que almacena datos de productos (**PRODUCTO.SEC**) según el registro declarado en la Fig. 3 y lo recorre completamente en forma secuencial (registro por registro) hasta encontrar el producto solicitado (Fig. 7).

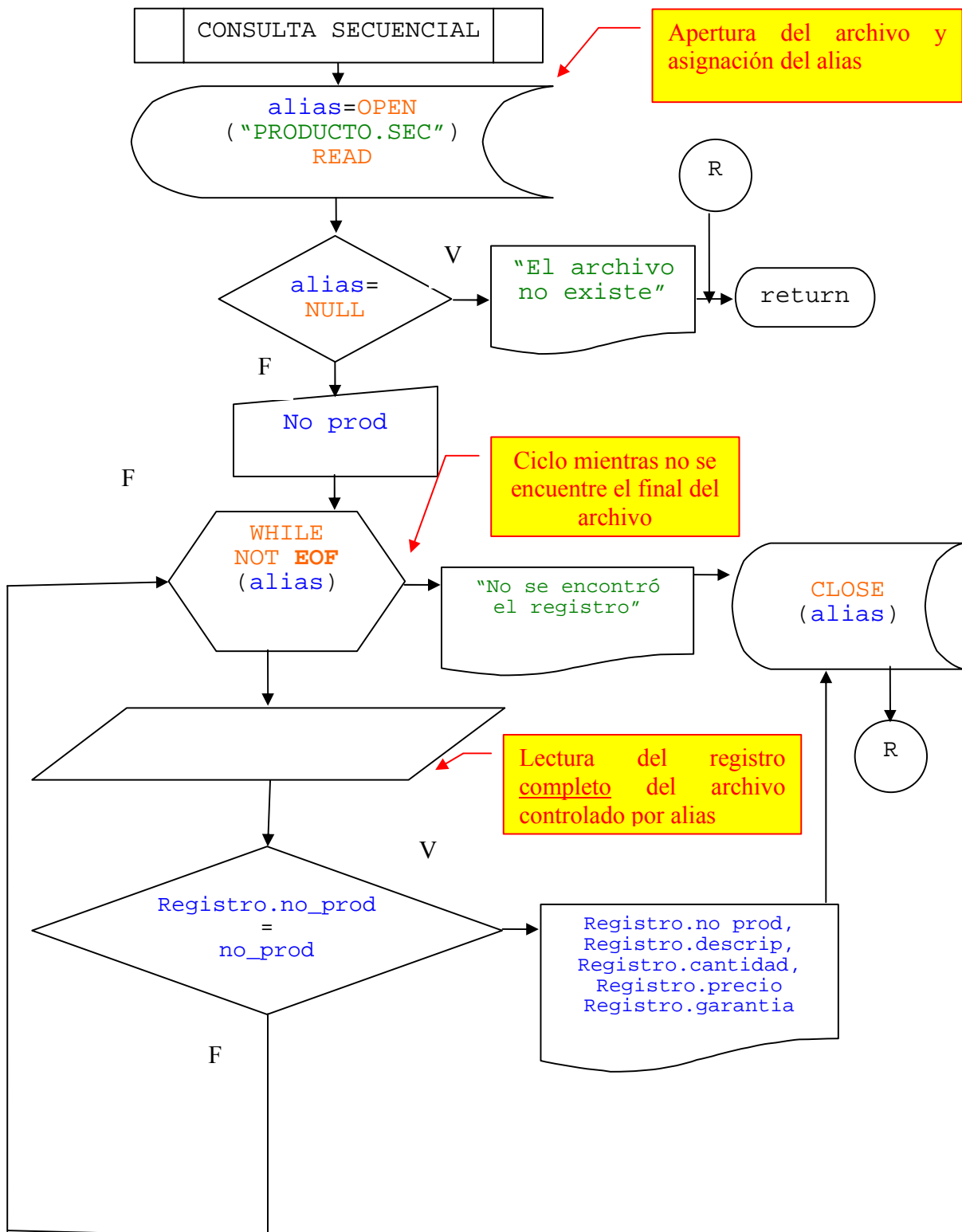


Fig. 9. Diagrama de flujo de rutina de consulta secuencial

## 1.7.2. Archivos directos (relativos, de acceso directo o aleatorios)

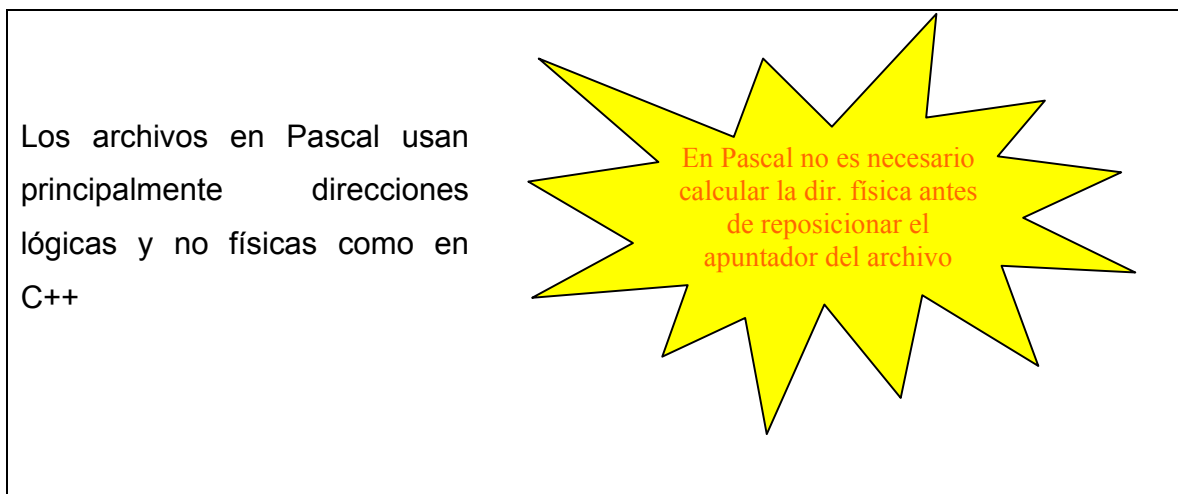
A diferencia de los archivos secuenciales, en los archivos directos no es necesario recorrerlo completamente para acceder un registro en particular, sino se puede colocar el apuntador interno del archivo directamente en el registro deseado, permitiendo con esto mayor rapidez de acceso. Cabe destacar que para reposicionar este apuntador se utiliza el comando `SEEK` indicándole la dirección del registro que se desea.

### 1.7.2.1. Direcciones lógicas y direcciones físicas

El lenguaje Pascal utiliza principalmente direcciones lógicas (aunque también maneja direcciones físicas) para los archivos (a diferencia de únicamente direcciones físicas de otros lenguajes como C++), esto es, que el direccionamiento consiste en el renglón que se asigna a un registro en lugar del espacio ocupado por los datos en el archivo (calculado en bytes). P. ejem. Suponga que tiene un archivo llamado “**PRODUCTO.SEC**” que almacena registros con el formato mostrado en la Fig. 3. Dicho registro tiene cinco campos, cada uno con un tipo de dato determinado y un tamaño específico (Fig.10).

Campo	Tipo de dato	Tamaño en bytes
no_prod	Integer	2
Descrip	Char [30]	30
Cantidad	Integer	2
Precio	Real	4
Garantia	Char	1
TOTAL		39

*Fig. 10. Ejemplo de cálculo del espacio ocupado por un registro*



*Fig.11. El lenguaje Pascal maneja archivos con direcciones lógicas*

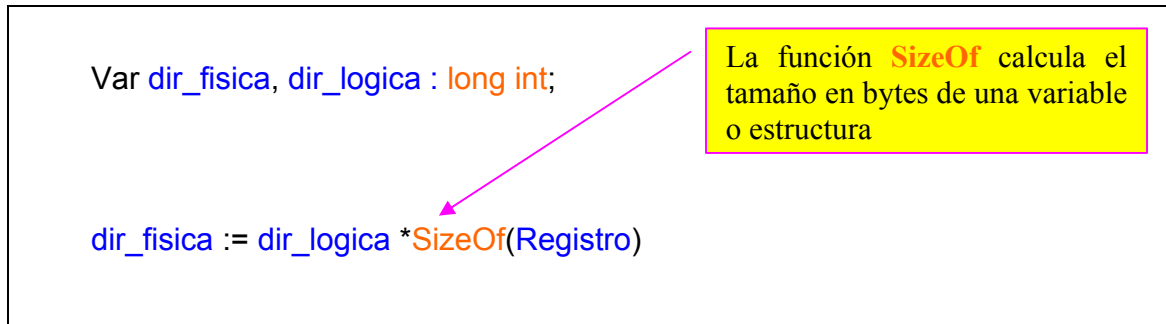
### 1.7.2.2. Control de direcciones lógicas

Para poder reposicionar el apuntador de un archivo en un registro específico en Pascal **NO** es necesario calcular su dirección física correspondiente de acuerdo al espacio ocupado por sus registros predecesores, sino únicamente conocer el renglón que ocupa.

Archivo: <b>PRODUCTO.SEC</b>						
Dir.	Dir.	No_prod	Descrip	Cantidad	Precio	Garantia
Lóg.	Fís.					
0	0	0	"Camisa de vestir"	100	65.80	'N'
1	41	1	"Pantalón para dama"	234	115.50	'N'
2	82	2	"Radiograbadora"	36	895.75	'S'
3	123	3	"Gabinete para sala"	54	1532.60	'N'
4	164					
5	205					
		Entero	Cadena [30]	Entero	Real	Caracter

*Fig. 12. Direcciones lógicas y físicas de un archivo*

Sin embargo, si se desea convertir direcciones lógicas en direcciones físicas se utiliza la fórmula mostrada en la Fig. 13.



*Fig. 13. Cálculo de la dirección física a partir de la dirección lógica*

### 1.7.2.3. Consulta directa

A continuación se muestra un diagrama de flujo de una rutina de consulta directa un registro en particular. En esta rutina se maneja el archivo **PRODUCTO.SEC** según el registro declarado en la Fig. 3. A diferencia del recorrido secuencial, en el que es necesario leer cada registro del archivo, en la consulta directa se posiciona el apuntador del archivo directamente en ese registro para leerlo (Fig. 14).

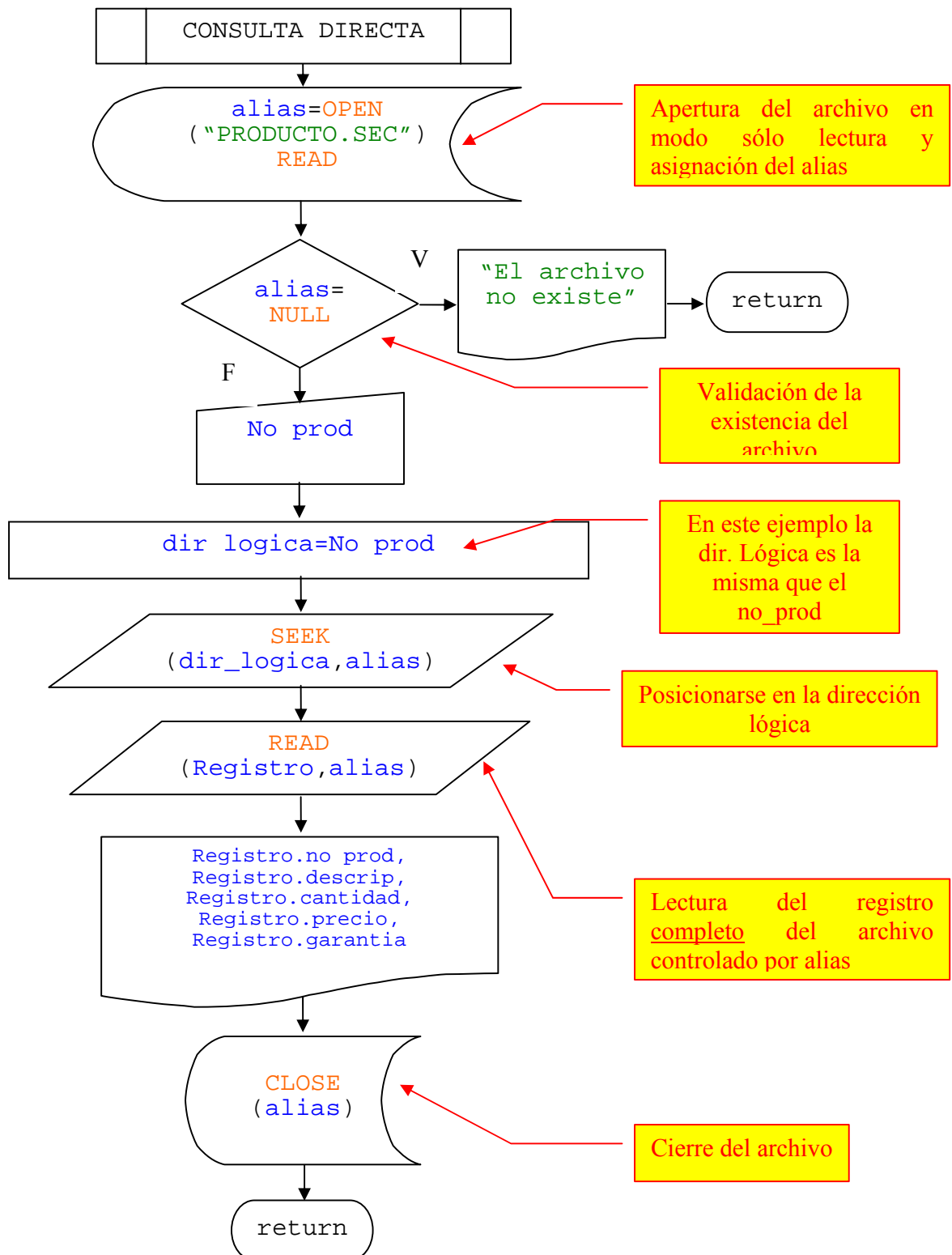


Fig. 14. Diagrama de flujo de rutina de consulta directa



## 2.FUNCIONES DE MANEJO DE ARCHIVOS EN PASCAL

Esta sección presenta Los aspectos generales de la implementación de programas de archivos en Pascal. Aunque se puede almacenar cualquier tipo de datos en archivos, aquí se muestran las operaciones del manejo de registros (*Record*) en archivos, por lo que las funciones que se mencionan a continuación son las mas importantes para este tipo de datos.

### 2.1. Declaración del alias del archivo

Para realizar programas de manejo de archivos en Pascal se requiere declarar una variable de tipo `FILE` que opere como el apuntador a la estructura del archivo (alias), indicando el tipo de contenido del archivo, esto se logra con la sig. línea:

```
Var alias : FILE OF Tipo_registro;
```

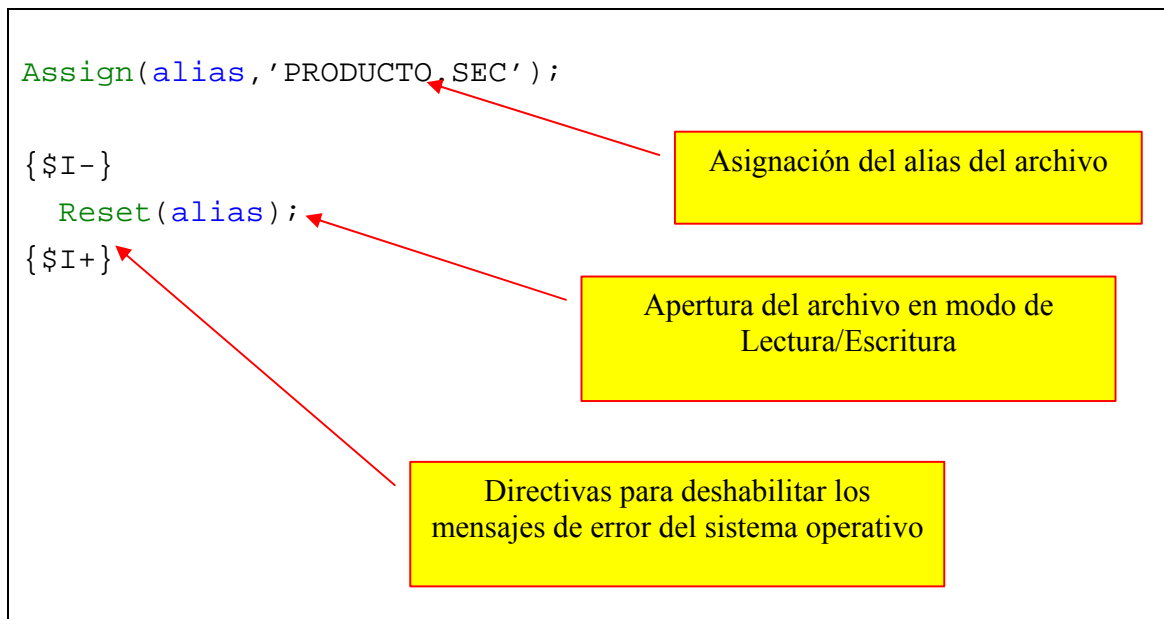
### 2.2. Funciones de manejo de archivos en Pascal

En esta sección se presentan las funciones más importantes para el manejo y control de registros en archivos.

#### 2.2.1 La función `Assign` y modos de apertura de archivos

Se usa la función `Assign` para asignar el alias de un archivo, es decir, se debe establecer una relación entre el nombre y la ubicación física del archivo con

un nombre corto conocido como alias. Una vez establecida esta relación, se realizan todas las operaciones en el archivo por medio de este. Para abrir un archivo en Pascal en modo de lectura y escritura se utiliza la función `Reset` y para crear un archivo se usa la función `Rewrite`.

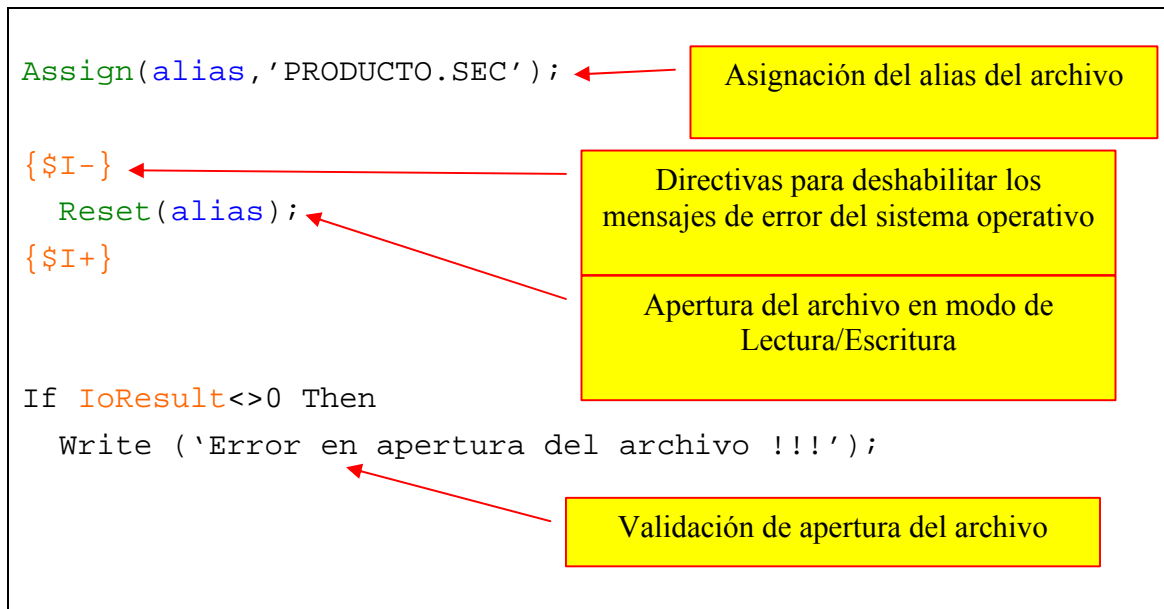


*Fig. 15. Las funciones `Assign` y `Reset`*

### 2.2.2. Validar la apertura de un archivo

Algunas funciones requieren la existencia del archivo para realizar operaciones, por ello es necesario verificar que cuando se intenta abrir un archivo haya tenido éxito la operación. Si un archivo no se puede abrir, la función `Reset` devuelve un valor diferente de 0 (cero). Para lograr esto, es necesario utilizar las directivas `{ $I- }` y `{ $I+ }`, las cuales deshabilitan y habilitan respectivamente los mensajes de error generados por el sistema operativo al fallar algún procedimiento. Cuando se invoca una función de este tipo, el resultado se almacena en la variable `IoResult`, la cual se utiliza para determinar si una función

tuvo éxito. La Fig. 16 muestra un ejemplo de la manera de detectar la apertura de un archivo.



*Fig. 16. Validar la apertura de un archivo*

### 2.2.3. Cierre de archivos usando `Close`

Antes de dejar de utilizar un archivo es necesario cerrarlo. Esto se logra mediante la función `Close` y es necesario indicarle el alias del archivo que se desea cerrar.

### 2.2.4. Escritura de registros usando `Write`

La función `Write` proporciona el mecanismo para almacenar **todos** los campos de un registro en un archivo. Cabe destacar que al utilizar esta función, se almacena una variable (de tipo `Record`) que representa un bloque de datos o campos; es decir, no se almacena campo por campo. Esta función tiene dos

argumentos: el alias del archivo donde se desea almacenar y la variable que se desea grabar (Fig. 17).

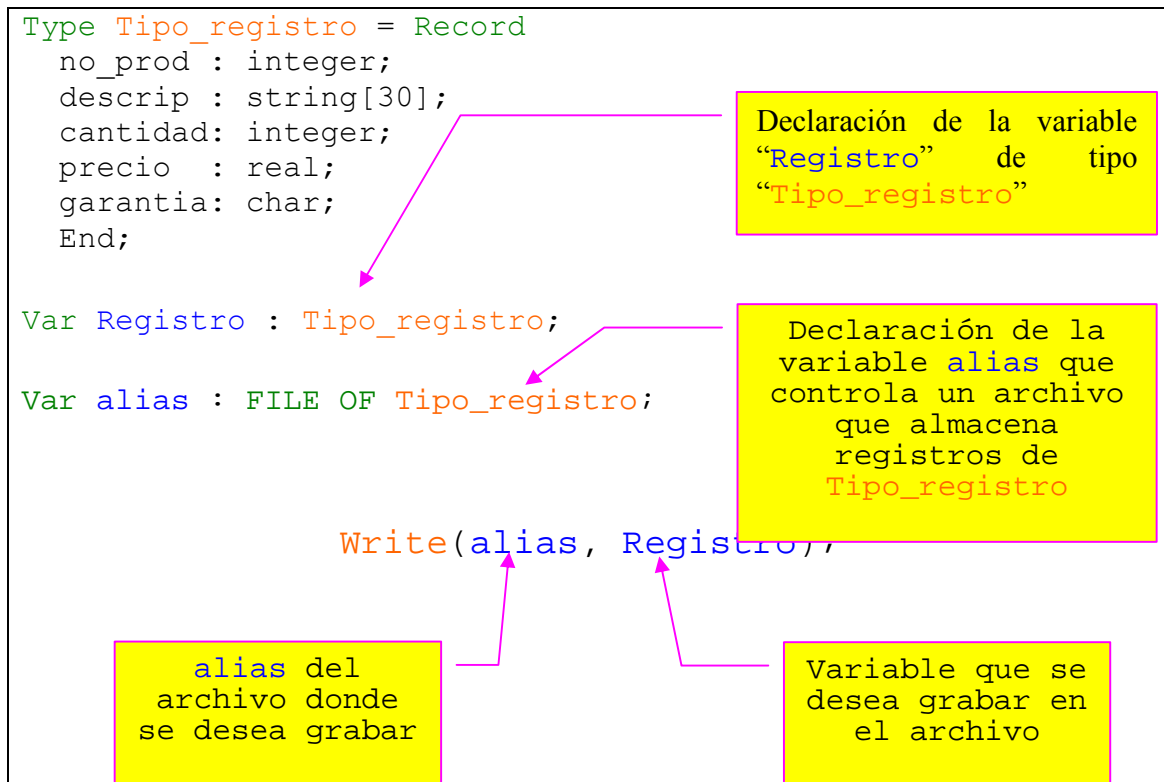


Fig. 17. La función Write

### 2.2.4.1. Vaciando los buffers con Flush

Un buffer es un área de almacenamiento temporal en memoria para el conjunto de datos leídos o escritos en el archivo. Estos buffers retienen datos en tránsito desde y hacia al archivo y tienen la finalidad de hacer más eficiente las operaciones de entrada/salida en los archivos de disco, provocando menor cantidad de accesos, los cuales son más lentos que la memoria. P. ejem. Si se requiere consultar constantemente un dato del archivo, no es necesario calcular su dirección física, reposicionar el apuntador del archivo, "cargar" el dato en memoria

mediante una operación de lectura cada vez que se necesita, sino que el sistema operativo controla y mantiene este dato en los buffers de memoria y de ahí lo toma cuando lo requiera. Sólo hay una consideración importante al utilizar los buffers, los datos escritos en ellos no se reflejan exactamente en los archivos de disco en forma inmediata, sino hasta que se “vacía” el buffer. Para ello se utiliza la función `Flush` y basta enviarle el alias del archivo como argumento. Los buffers también se vacían cuando se cierra el archivo.

### 2.2.5. Lectura de registros usando `Read`

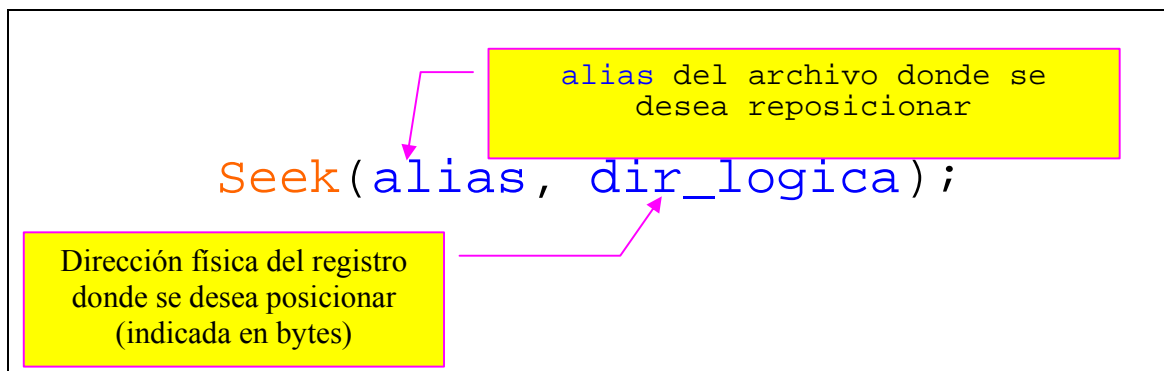
La función `Read` permite “cargar” **todos** los campos de un registro en un archivo, es decir, lee un registro y lo copia en la memoria RAM (Fig. 1). Esta función tiene los mismos argumentos que la función `Write` (Fig. 17).

### 2.2.6. Reposicionando el apuntador mediante `Seek`

Para comprender la operación de esta función es necesario estar relacionado con el apuntador del archivo, el cual indica la posición dentro del archivo. Cuando se abre un archivo en modo de sólo lectura, sólo escritura o lectura/escritura, el apuntador del archivo se posiciona al inicio del mismo y cuando un archivo se abre en modo agregar se posiciona al final, sin embargo, se puede reposicionar este apuntador del archivo mediante la función `Seek`. También es importante mencionar que cada vez que se realiza una operación de lectura o de escritura de cualquier tipo de datos (caracter, cadena, estructura, etc.), el apuntador del archivo se mueve al final de dicho dato, de tal forma que está posicionado en el siguiente, por lo que es muy importante asegurarse que se encuentre en la posición deseada antes de realizar cualquier operación.

Como se mencionó en las secciones 1.7.2.1. y 1.7.2.2., los archivos en Pascal son controlados por direcciones lógicas (no físicas) y la función `Seek` permite

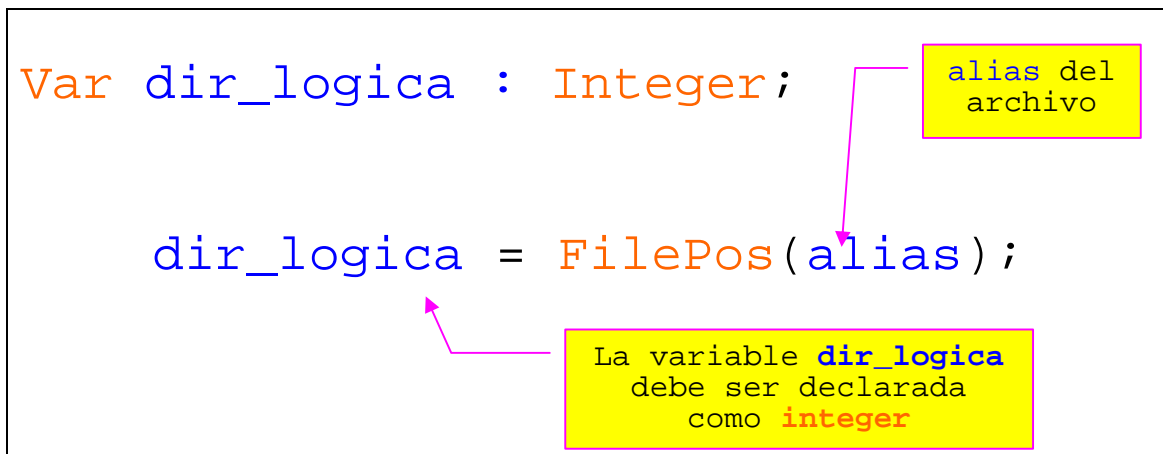
reposicionar el apuntador del archivo en la dirección deseada mediante dos argumentos: el alias del archivo y la dirección lógica. Para poder reposicionar el apuntador del archivo es necesario que éste se encuentre abierto y se le haya asignado el **alias** correspondiente. Es muy importante destacar que debe asegurarse que la dirección lógica tenga un valor dentro del rango permitido, es decir, que no exceda el límite del archivo. La Fig. 18 ilustra el uso de esta función.



*Fig. 18. La función Seek.*

#### 2.2.6.1. Conociendo la posición del apuntador del archivo con la función FilePos

Se usa la función `FilePos` para conocer la posición actual del apuntador de un archivo abierto. La posición se expresa mediante un valor entero que representa los renglones (dirección lógica) contados desde el principio del archivo (Fig. 19).



*Fig. 19. La función FilePos*

### 2.2.7. Detectando el final del archivo con Eof

Se usa el macro `Eof` para determinar si se ha encontrado el final de un archivo. Si se encuentra el final de un archivo, devuelve un valor diferente de cero y cero en caso contrario. Para invocarlo es necesario colocar el alias del archivo abierto como argumento. Se utiliza mucho esta función en recorridos de tipo secuencial (Ver sección 1.7.1).

### 2.2.8. Cambiando nombres de archivos mediante Rename

Esta función tiene como objetivo cambiar el nombre de un archivo o subdirectorio especificado por su ruta de acceso. Sólo necesita dos argumentos: el nombre anterior del archivo y el nuevo nombre. **Es importante destacar que esta función sólo puede aplicarse a archivos cerrados.**

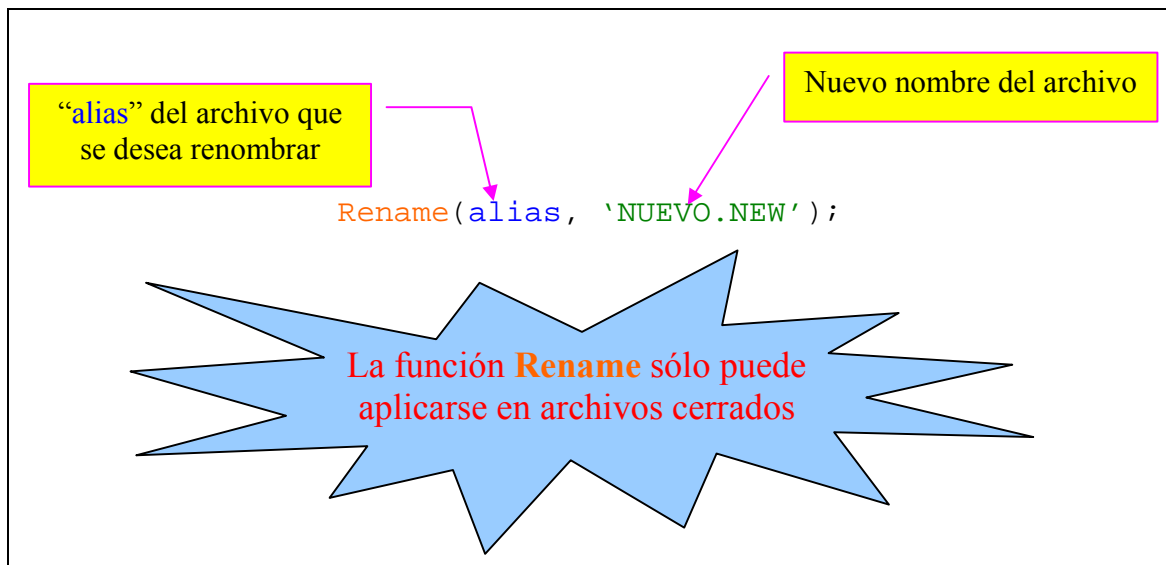


Fig. 20. La función *Rename*

### 2.2.9. Eliminando archivos con la función **Erase**

La función `Erase` elimina definitivamente un archivo especificando su nombre (Fig. 21). **Es importante destacar que esta función sólo puede aplicarse a archivos cerrados.**

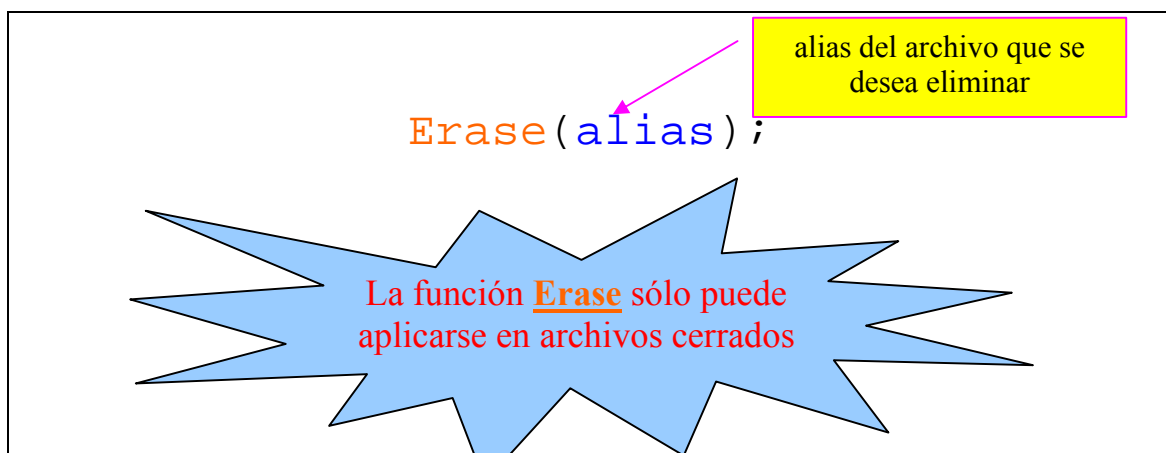


Fig. 21. La función *Erase*



## 3. APLICACIONES DE ARCHIVOS EN PASCAL

A continuación se presentan algunas de las aplicaciones prácticas más comunes con archivos. Están basadas en los aspectos teóricos analizados en la sección 1 y en las funciones de manejo de archivos en Pascal de la segunda sección. Aquí se analizarán los procedimientos de inserción, consulta y eliminación de registros tanto en archivos secuenciales como directos, así como el listado secuencial. En ambos tipos de archivos se tratarán como archivos binarios.

Para apoyar el análisis, diseño e implementación de estas rutinas, se tomará el ejemplo del control de PRODUCTOS que se mencionó anteriormente así como la definición del registro de la Fig. 3.

En cada procedimiento se muestra el diagrama de flujo y la codificación íntegra en Pascal.

### 3.1. Declaraciones globales

Antes de analizar, diseñar e implementar las rutinas es necesario establecer las consideraciones iniciales tales como las declaraciones globales del registro de productos y el alias correspondiente. La Fig. 22 muestra las declaraciones globales necesarias para estas aplicaciones. Para efectos de identificación, se usará el nombre “PRODUCTO.SEC” para el archivo secuencial y el nombre “PRODUCTO.DIR” para el archivo directo.

```

Program Arch_Sec;
Uses CRT;

Type Tipo_registro = Record   { Campos:  }
  no_prod : integer;           { Numero de producto}
  descrip : string[30];        { Descripcion del articulo}
  cantidad: integer;           { Cantidad disponible en almacen}
  precio  : real;              { Precio del articulo}
  garantia: char;              { Tiene garantia? [S/N]}
End;

Var Registro : Tipo_registro;   { Declaracion global de la variable
                                "Registro" de tipo struct
tipo_registro}

    alias      : FILE OF Tipo_registro; { Declaracion global de la variable
                                "alias" (apuntador a un archivo)}

```

*Fig. 22. Declaraciones globales de las aplicaciones*

## 3.2. Archivos Secuenciales en Pascal

En esta sección se analizará la manera de diseñar rutinas que manipulen registros de productos o artículos en un archivo secuencial. Como su nombre lo indica, en este tipo de archivo se hace un recorrido secuencial para localizar la dirección del registro solicitado, es decir, se lee registro por registro hasta llegar al deseado.

### 3.2.1. Altas Secuenciales

Aquí se presenta una rutina que inserta registros de productos en un archivo secuencial. Se considera un número de producto (campo `no_prod`) que servirá como referencia para identificarlo y diferenciarlo de otros productos y no se permite que se graben dos productos diferentes con el mismo número, por lo que es necesario realizar un recorrido secuencial completo del archivo para asegurarse que no haya duplicados.

La primera ocasión que se intente insertar registros en un archivo, éste debe crearse; sin embargo **debe cuidarse no crear el archivo cada vez que se**

invoque esta rutina porque debe tenerse presente que si se crea un archivo existente, se pierde su contenido anterior.

### **3.2.1.1. Diagrama de flujo de la rutina de Altas Secuenciales**

La Fig. 23 ilustra el diagrama de flujo de la rutina de altas en un archivo secuencial.

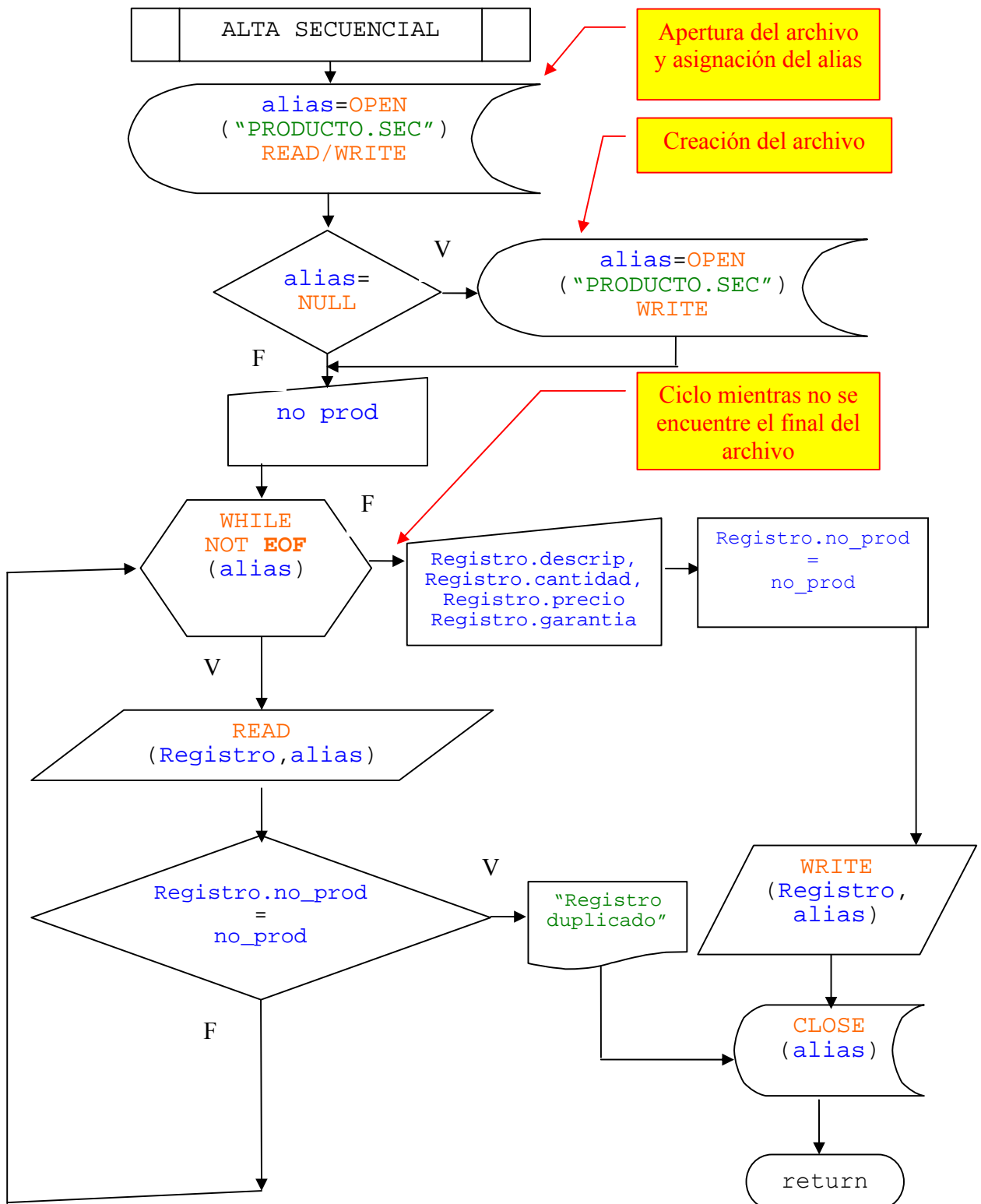


Fig. 23. Diagrama de flujo de rutina de alta secuencial

### 3.2.1.2. Codificación de la rutina de ALTAS secuenciales

La Fig. 24 muestra el código de la rutina de ALTAS secuenciales de acuerdo al diagrama de la Fig. 23.

```

Procedure ALTA_SECUENCIAL;
Var no_prod: integer; { Variable local para el numero de producto}
Begin
  clrscr;
  writeln('ALTAS DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
{$I+}
  If(IoResult<>0) Then
    Rewrite(alias); { Crea el archivo en caso de no existir }

  write('Numero de producto: '); readln(no_prod);

  While(Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
  Begin
    Read(alias,Registro);
    { Lee el "Registro", de tamano=sizeof(Registro) del archivo
"alias" }
    If(Registro.no_prod=no_prod) Then
      Begin
        writeln('Registro duplicado !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Close(alias);
        Readkey;
        Exit;
      End;
    End;
  End;
  write('Descripcion: '); readln(Registro.descripcion);
  write('Cantidad : '); readln(Registro.cantidad);
  write('Precio : '); readln(Registro.precio);
  Repeat
    write('Garantia : '); readln(Registro.garantia);
    Registro.garantia:=Ucase(Registro.garantia);
  Until(Registro.garantia='S') OR (Registro.garantia='N');
  Registro.no_prod:=no_prod;
  Write(alias,Registro); { Grabar el Registro completo }
  Close(alias); { Cierra el archivo }
  writeln('Producto registrado !!!');
  writeln('<<< Oprima cualquier tecla para continuar >>>');
  readkey;
End;

```

*Fig. 24. Codificación de la rutina de altas secuenciales*

### 3.2.2. CONSULTAS secuenciales

En esta sección se analiza una rutina que busca un registro particular de un producto en un archivo secuencial. En este tipo de archivo se inicia el recorrido desde el primer registro almacenado y se detiene cuando se localiza el registro solicitado o cuando se encuentre el final del archivo.

#### 3.2.2.1. Diagrama de flujo de la rutina de CONSULTAS secuenciales

El diagrama de flujo de esta rutina se planteó en la sección 1.7.1.1 y se muestra en la Fig. 9.

#### 3.3.2.2. Codificación de la rutina de CONSULTAS secuenciales

La Fig. 25 muestra el código de la rutina de CONSULTAS secuenciales de acuerdo al diagrama de la Fig. 7.

```
Procedure CONSULTA_SECUENCIAL;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
Begin
  Clrscr;

  writeln('CONSULTA DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
{$I+}

  If(IoResult<>0) Then
  Begin
```

```

        writeln('No existe el archivo !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Readkey;
        Exit;
    End;

    write('Numero de producto: '); Read(no_prod);

    While (Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
    Begin
        Read(alias, Registro);
        { Lee el "Registro", de tamaño=sizeof(Registro) del archivo
"alias" }
        If (Registro.no_prod=no_prod) Then
        Begin
            writeln('No Prod                Descripcion  Cantidad
Precio  Garantia');
            writeln('-----
-----');

            writeln(Registro.no_prod:3, Registro.descrip:30, Registro.cantidad:3, '
', Registro.precio:4:2, ' ', Registro.garantia);
            writeln('<<< Oprima cualquier tecla para continuar >>>');
            Close(alias);
            Readkey;
            Exit;
        End;
    End;

    writeln('No se encuentra ese registro !!!');
    Close(alias); { Cierra el archivo }
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
End;

```

*Fig. 25. Codificación de la rutina de consultas secuenciales*

### 3.2.3. LISTADO secuencial

Esta rutina es muy semejante a la de CONSULTAS secuenciales, solo que en el LISTADO se despliegan todos los registros lógicos que contiene el archivo. En este tipo de archivo se inicia el recorrido desde el primer registro almacenado y se detiene cuando se encuentre el final del archivo.

### 3.2.3.1. Diagrama de flujo de la rutina de LISTADO secuencial

El diagrama de flujo de esta rutina se muestra en la Fig. 26.

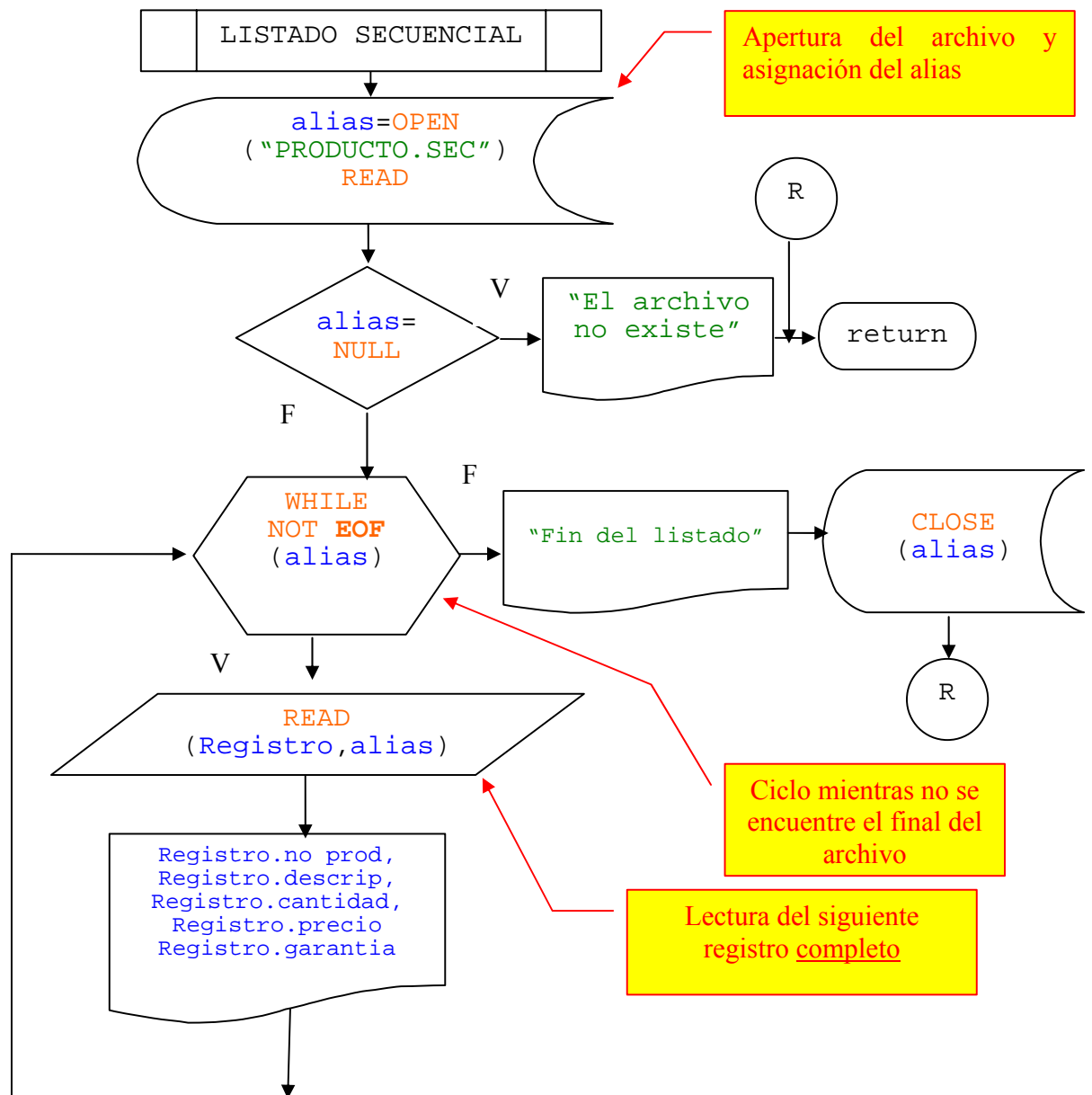


Fig. 26. Diagrama de flujo de rutina de listado secuencial



### 3.3.3.2. Codificación de la rutina de LISTADO secuencial

La Fig. 27 muestra el código de la rutina de LISTADO secuencial de acuerdo al diagrama de la Fig. 26.

```

Procedure LISTADO_SECUENCIAL;
Begin

  Clrscr;
  writeln('LISTADO DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
{$I+}

  If(IoResult<>0) Then
  Begin
    writeln('No existe el archivo !!!');
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
    Exit;
  End;

  writeln('No Prod          Descripcion  Cantidad          Precio
Garantia');
  writeln('-----');
  writeln('-----');

  While(Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
  Begin
    Read(alias,Registro);
    { Lee el "Registro", de tamano=sizeof(Registro) del archivo
"alias" }
    writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,'
',Registro.precio:4:2,' ',Registro.garantia);
  End;

  writeln('Fin del listado !!!');
  Close(alias); { Cierra el archivo }
  writeln('<<< Oprima cualquier tecla para continuar >>>');
  Readkey;
End;

```

*Fig. 27. Codificación de la rutina de listado secuencial*

### **3.2.4. MODIFICACIONES de datos en un archivo secuencial**

La forma de modificar el contenido de los campos de registros de un archivo, depende mucho de la rutina de consulta, ya que es necesario localizar previamente el registro que se desea modificar, capturar los nuevos valores y posteriormente **grabar el registro completo en la misma posición que se encontraba.** Esto último es muy importante porque recuerde que cuando se termina de leer un registro del archivo, el apuntador se posiciona al inicio del siguiente registro, por lo que, antes de grabar el registro modificado, es necesario reposicionar el apuntador del archivo en la dirección correcta.

#### **3.2.4.1. Diagrama de flujo de la rutina de MODIFICACIÓN secuencial**

La Fig. 28 muestra el diagrama de flujo de la rutina de modificaciones de campos de un registro particular en un archivo secuencial.

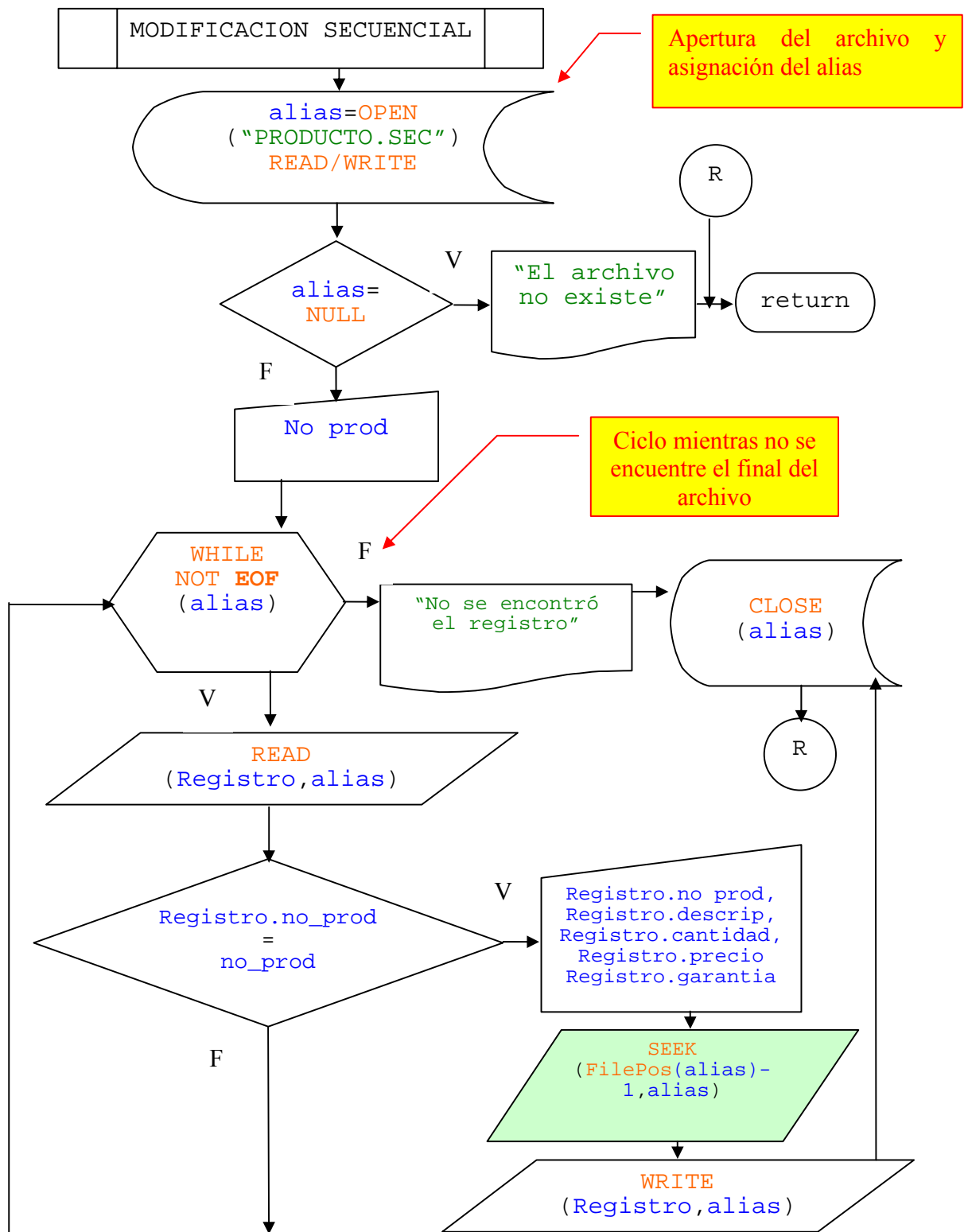


Fig. 28. Diagrama de flujo de rutina de modificación secuencial

### 3.2.4.2. Codificación de la rutina de MODIFICACIÓN secuencial

La Fig. 29 muestra la codificación de la rutina de modificaciones de campos de un registro particular en un archivo secuencial.

```

Procedure MODIFICACION_SECUENCIAL;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
Begin

  Clrscr;
  writeln('MODIFICACION DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
  {$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
  {$I+}

  If(IoResult<>0) Then
  Begin
    writeln('No existe el archivo !!!');
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
    Exit;
  End;

  write('Numero de producto: '); Read(no_prod);

  While(Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
  Begin
    Read(alias,Registro);
    { Lee el "Registro", de tamaño=sizeof(Registro) del archivo
"alias" }
    If(Registro.no_prod=no_prod) Then
    Begin
      writeln('No Prod          Descripcion  Cantidad
Precio  Garantia');
      writeln('-----
-----');

      writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3, '
',Registro.precio:4:2, '    ',Registro.garantia);

      writeln('Anote los nuevos datos ...');
      write('Descripcion: '); Readln(Registro.descrip);
      write('Cantidad   : '); Readln(Registro.cantidad);
      write('Precio     : '); Readln(Registro.precio);
    End;
  End;

```

```

Repeat
  write('Garantia   : '); readln(Registro.garantia);
  Registro.garantia:=Ucase(Registro.garantia);
  Until(Registro.garantia='S') OR (Registro.garantia='N');

{ Es necesario reposicionar el apuntador del archivo al principio
del
registro que desea modificar, ya que al leer un registro, el
apuntador se posiciona en el registro siguiente
La funcion FilePos(alias) devuelve la posicion donde se encuentra
el
apuntador }
Seek(alias,FilePos(alias)-1);
Write(alias,Registro); { Graba el registro con los nuevos campos}
Close(alias); { Cierra el archivo }
writeln('Registro modificado !!!');

writeln('<<< Oprima cualquier tecla para continuar >>>');
Readkey;
Exit;
End;
End;

writeln('No se encuentra ese registro !!!');
Close(alias); { Cierra el archivo }
writeln('<<< Oprima cualquier tecla para continuar >>>');
Readkey;
End;

```

*Fig. 29. Codificación de rutina de modificación secuencial*

### 3.2.5. Bajas de registros en un archivo secuencial (bajas lógicas y bajas físicas)

Básicamente existen dos formas de eliminar registros en un archivo: lógica y físicamente. La diferencia radica en que cuando se elimina un registro en forma lógica de un archivo, sólo se le coloca una marca especial en alguno de sus campos que lo identifique como registro “borrado”, sin embargo, el registro sigue existiendo en el archivo y por lo tanto ocupa espacio. En cambio en la baja física de registros, se elabora una rutina que elimine físicamente el registro del archivo, es decir, que el archivo solo conserve los registros válidos. Las bajas físicas también son conocidas como rutinas de compactación o compresión del archivo.

### **3.2.5.1. Diagrama de flujo de la rutina de BAJAS lógicas en un archivo secuencial**

Como se mencionó en el punto anterior, las bajas lógicas consisten en “marcar” los registros eliminados. En el ejemplo práctico que se muestra a continuación, el registro borrado se “limpia”, dejando en blanco todos sus campos (colocando el valor cero en los campos numéricos y blancos en las cadenas o de tipo carácter).

Para fines prácticos, la rutina de bajas lógicas se asemeja mucho a la rutina de modificaciones, sólo que en las bajas no se capturan los nuevos valores, sino se les asigna valores en blanco y se graba el registro completo en la misma posición en la que se encontraba. El diagrama de la Fig. 30 muestra esta rutina.

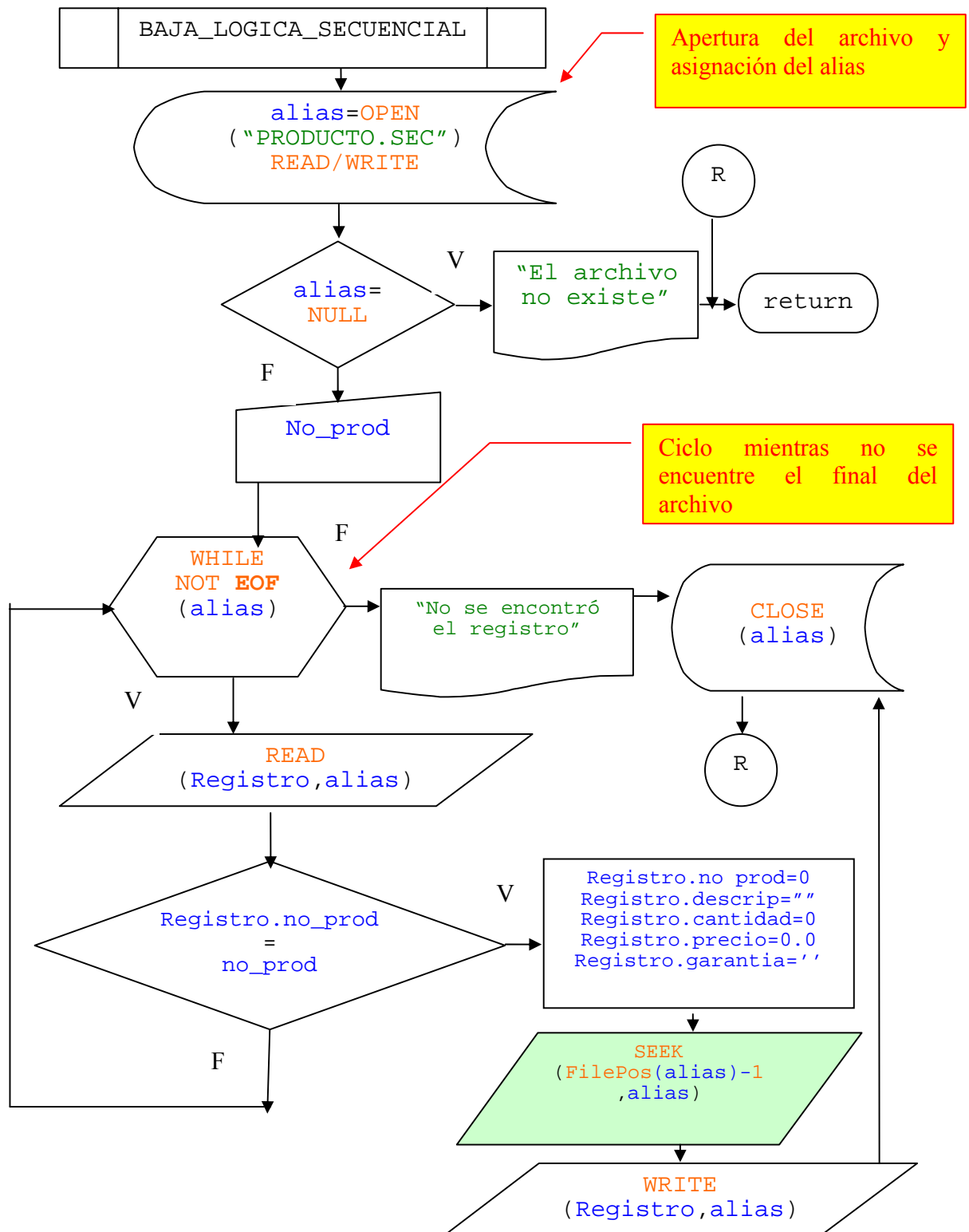


Fig. 30. Diagrama de flujo de rutina de baja lógica secuencial

### 3.2.5.2. Codificación de la rutina de BAJAS lógicas en un archivo secuencial

La Fig. 31 muestra la codificación íntegra de esta rutina.

```

Procedure BAJA_LOGICA_SECUENCIAL;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
    op      : char;
Begin
    Clrscr;
    writeln('BAJAS LOGICAS DE REGISTROS DE PRODUCTOS');

    Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
    {$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
    Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
    {$I+}

    If(IoResult<>0) Then
    Begin
        writeln('No existe el archivo !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Readkey;
        Exit;
    End;

    write('Numero de producto: '); Read(no_prod);

    While(Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
    Begin
        Read(alias,Registro);
        { Lee el "Registro", de tamaño=sizeof(Registro) del archivo
"alias" }
        If(Registro.no_prod=no_prod) Then
        Begin
            writeln('No Prod          Descripcion  Cantidad
Precio  Garantia');
            writeln('-----');
            writeln('-----');

            writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,
',Registro.precio:4:2,', ' ',Registro.garantia);

            Registro.no_prod:=0;
            Registro.descrip:='';
            Registro.cantidad:=0;
            Registro.precio:=0.0;
            Registro.garantia:=' ';
        End;
    End;
End;

```



```

Repeat
  write('Esta seguro que desea borrarlo? '); readln(op);
  op:=Uppcase(op);
Until (op='S') OR (op='N');

{ Es necesario reposicionar el apuntador del archivo al principio
del
registro que desea modificar, ya que al leer un registro, el
apuntador se posiciona en el registro siguiente
La funcion FilePos(alias) devuelve la posicion donde se encuentra
el
apuntador }
If (op='S') Then
Begin
  Seek(alias,FilePos(alias)-1);
  Write(alias,Registro); { Graba el registro con los nuevos
campos}
  Close(alias); { Cierra el archivo }
  writeln('Registro eliminado logicamente !!!');
End;

  writeln('<<< Oprima cualquier tecla para continuar >>>');
  Readkey;
  Exit;
End;
End;

writeln('No se encuentra ese registro !!!');
Close(alias); { Cierra el archivo }
writeln('<<< Oprima cualquier tecla para continuar >>>');
Readkey;
End;

```

*Fig. 31. Codificación de rutina de baja lógica secuencial*

### 3.2.5.3. Diagrama de flujo de la rutina de BAJAS físicas en un archivo secuencial (compactar)

La rutina de bajas físicas (también conocida como compactar el archivo), consiste en eliminar definitivamente los espacios dejados por los registros borrados lógicamente. Cuando se elimina un registro en forma lógica dejando en blanco sus campos, sigue ocupando espacio en el archivo, sin embargo se puede diseñar una rutina que se apoye de un archivo auxiliar (también secuencial), en el cual se graben todos los registros válidos, es decir, los registros no eliminados,

para posteriormente eliminar el archivo original y renombrar este archivo auxiliar como el archivo original. La Fig. 32 muestra este diagrama de flujo.

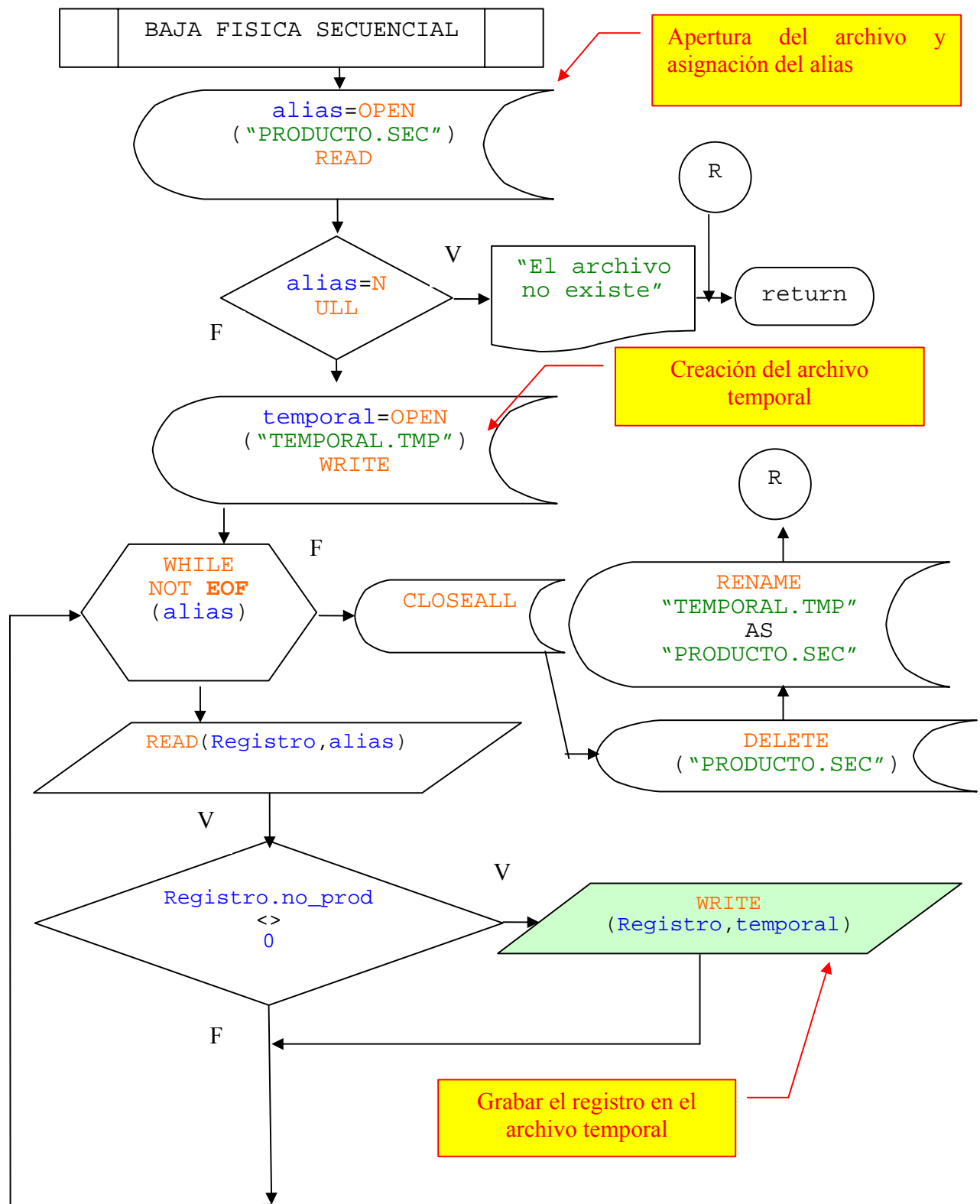


Fig. 32. Diagrama de flujo de rutina de baja física secuencial (compactar)

### 3.2.5.4. Codificación de la rutina de BAJAS físicas en un archivo secuencial (compactar)

La Fig. 33 muestra la codificación íntegra de esta rutina.

```

Procedure BAJA_FISICA_SECUENCIAL;
Var temporal : FILE OF Tipo_registro;
Begin

    Clrscr;
    writeln('BAJA FISICA DE REGISTROS DE PRODUCTOS');

    Assign(alias,'PRODUCTO.SEC'); { Asignacion del nombre del archivo al
alias}
    {$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
    Reset(alias); {Intenta abrir el archivo PRODUCTO.SEC
en modo de lectura/escritura}
    {$I+}

    If(IoResult<>0) Then
    Begin
        writeln('No existe el archivo !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Readkey;
        Exit;
    End;

    Assign(temporal,'TEMPORAL.TMP'); { Asigna el alias del archivo auxiliar
}
    Rewrite(temporal); { Crea el archivo TEMPORAL.TMP }

    writeln('No Prod          Descripcion  Cantidad          Precio
Garantia');
    writeln('-----');
    writeln('-----');

    While(Not Eof(alias)) Do { Ciclo mientras no se encuentre el final del
archivo}
    Begin
        Read(alias,Registro);
        { Lee el "Registro", de tamano=sizeof(Registro) del archivo
"alias" }
        If(Registro.no_prod<>0) Then
        Begin

writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,'
',Registro.precio:4:2,' ',Registro.garantia);
            Write(temporal,Registro);
        End;
    End;

    Close(temporal);

```

```
Close(alias); { Cierra los archivos }
Erase(alias); { Elimina el archivo PRODUCTO.SEC }
Rename(temporal,'PRODUCTO.SEC'); { Renombra el archivo TEMPORAL.TMP
como
                                PRODUCTO.SEC }
writeln('Archivo compactado !!!');
writeln('<<< Oprima cualquier tecla para continuar >>>');
Readkey;
End;
```

*Fig. 33. Codificación de rutina de baja física secuencial (compactar)*

### 3.3. Archivos directos en Pascal

En esta sección se analizará la manera de diseñar rutinas que manipulen registros de productos o artículos en un archivo directo. A diferencia del archivo secuencial en el que se hace un recorrido secuencial para localizar la dirección del registro solicitado, en el archivo de acceso directo, se posiciona el apuntador del archivo directamente en el registro solicitado usando la función `Seek`. Sin embargo **es muy importante destacar que esta función permite posicionar el apuntador interno del archivo en una dirección lógica válida**; esto es, que marcará error al intentar posicionarse fuera del archivo mediante una dirección lógica que exceda el tamaño del mismo. P. ejem. En la rutina de altas directas, debe asegurarse la existencia de la dirección en la que se desea insertar el nuevo registro y, en caso, que exceda el límite del archivo, deben incrustarse registros vacíos antes del nuevo registro. La Fig. 34 ilustra esto.

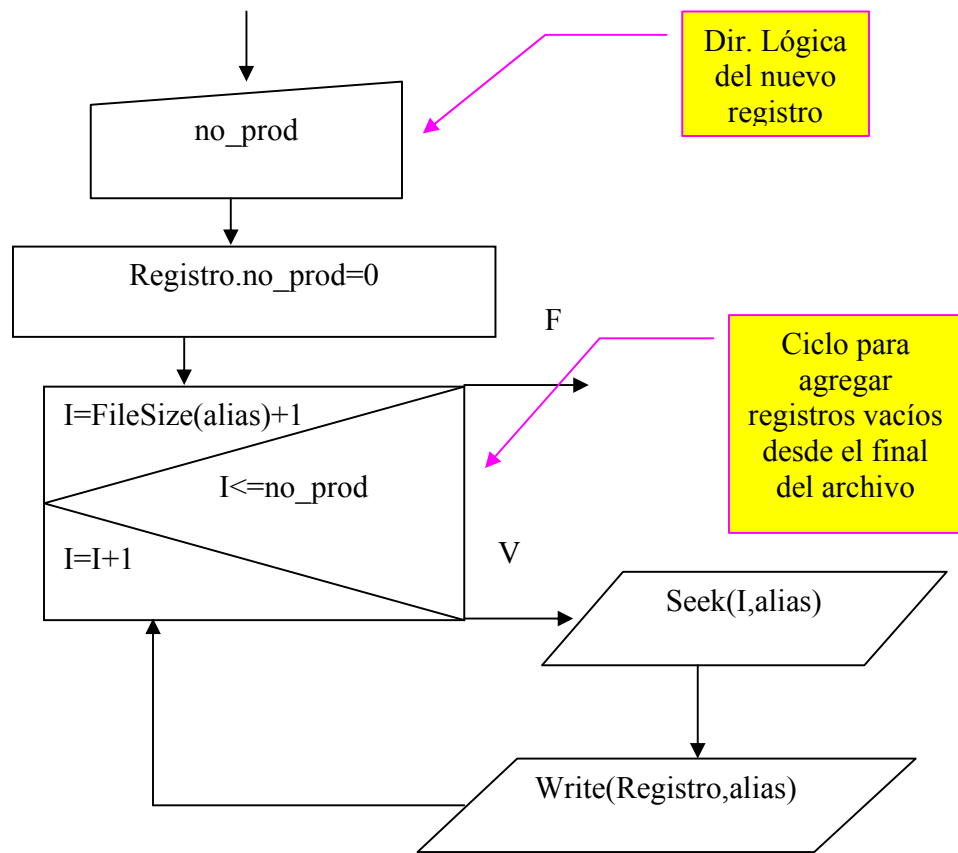


Fig. 34. Inserción de registros en blanco desde el final del archivo

### 3.3.1. Altas Directas

Aquí se presenta una rutina que inserta registros de productos en un archivo directo. Se tomará el mismo ejemplo del registro de producto que se usó en el archivo secuencial. En este caso se nombrará el archivo como **“PRODUCTO.DIR”**

La primera ocasión que se intente insertar registros en un archivo, éste debe crearse; sin embargo **debe cuidarse no crear el archivo cada vez que se invoque esta rutina** porque debe tenerse presente que si se crea un archivo existente, se pierde su contenido anterior. También debe considerarse no intentar

reposicionar el apuntador del archivo fuera del mismo (no exceder el tamaño del archivo).

### **3.3.1.1. Diagrama de flujo de la rutina de Altas directas**

La Fig. 35 ilustra el diagrama de flujo de la rutina de altas en un archivo relativo o de acceso directo.

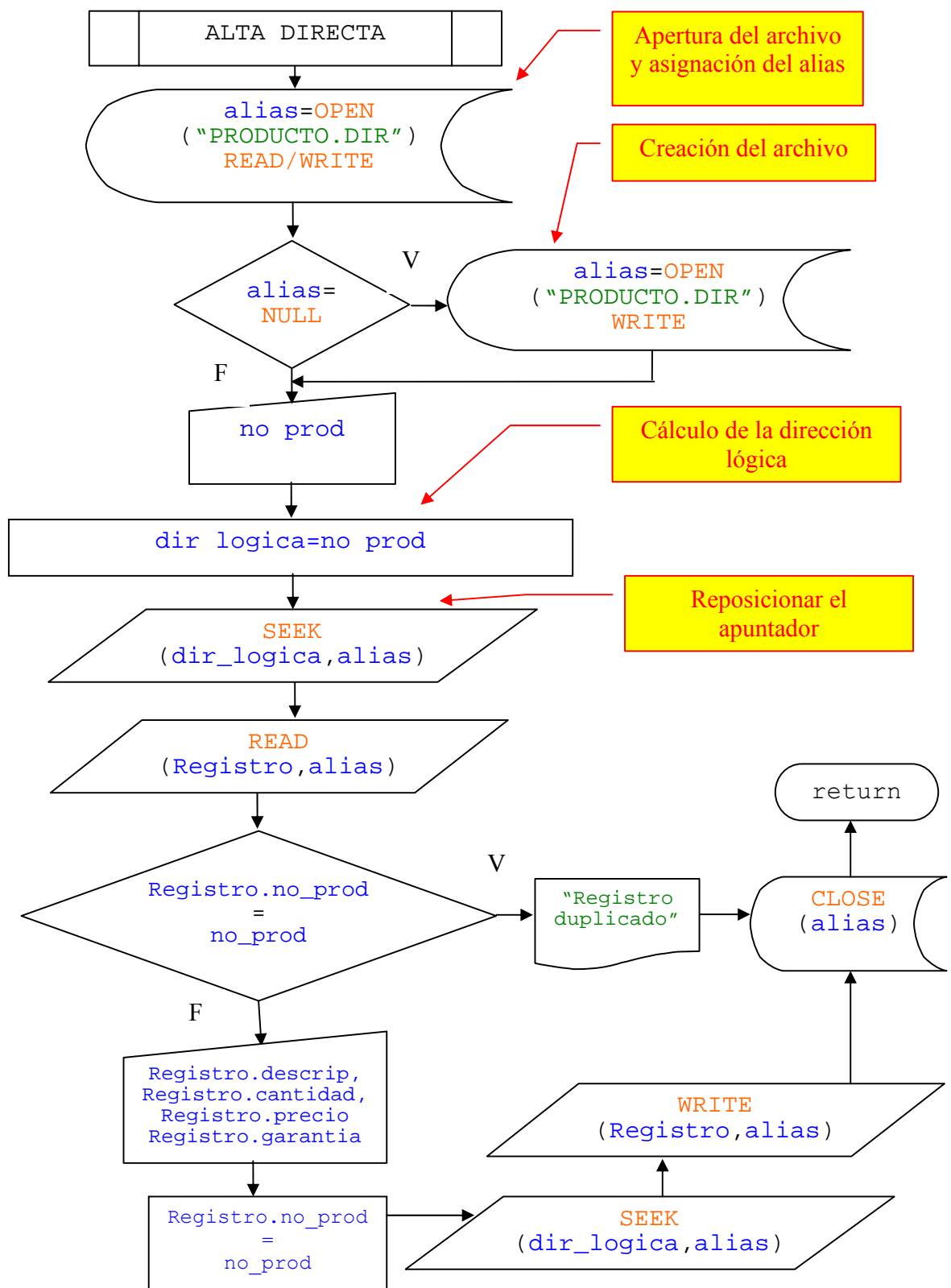


Fig. 35. Diagrama de flujo de rutina de altas directas



### 3.3.1.2. Codificación de la rutina de Altas directas

La Fig. 36 muestra la codificación de la rutina de altas en un archivo relativo o de acceso directo.

```

Procedure ALTA_DIRECTA;
Var no_prod : Integer; { Variable local para el numero de producto}
    i       : Integer;
Begin
    clrscr;
    writeln('ALTAS DE REGISTROS DE PRODUCTOS');

    Assign(alias,'PRODUCTO.DIR'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
    Reset(alias); {Intenta abrir el archivo PRODUCTO.DIR
                  en modo de lectura/escritura}
{$I+}

    If(IoResult<>0) Then
        Rewrite(alias); { Crea el archivo en caso de no existir }

    write('Numero de producto: '); readln(no_prod);

    Registro.no_prod:=0;
    For i:=FileSize(alias)+1 to no_prod Do
    Begin
        Seek(alias,i);
        Write(alias,Registro);
    End;

    Seek(alias,no_prod);
    Read(alias,Registro);

    { Lee el "Registro", de tamaño=sizeof(Registro) del archivo "alias" }
    If(Registro.no_prod=no_prod) Then
    Begin
        writeln('Registro duplicado !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Close(alias);
        Readkey;
        Exit;
    End;

    write('Descripcion: '); readln(Registro.descripcion);
    write('Cantidad    : '); readln(Registro.cantidad);
    write('Precio      : '); readln(Registro.precio);
    Repeat
        write('Garantia    : '); readln(Registro.garantia);
        Registro.garantia:=Ucase(Registro.garantia);
    Until(Registro.garantia='S') OR (Registro.garantia='N');

    Registro.no_prod:=no_prod;

```

```
Seek(alias,Registro.no_prod);  
Write(alias,Registro); { Grabar el Registro completo }  
Close(alias); { Cierra el archivo }  
  
writeln('Producto registrado !!!');  
writeln('<<< Oprima cualquier tecla para continuar >>>');  
readkey;  
End;
```

*Fig. 36. Codificación de rutina de altas directas*

### **3.3.2. CONSULTAS directas**

En esta sección se analiza una rutina que busca un registro particular de un producto en un archivo directo. En este tipo de archivo no es necesario hacer el recorrido secuencial desde el primer registro almacenado, sino se posiciona el apuntador del archivo directamente en la dirección lógica deseada.

#### **3.3.2.1. Diagrama de flujo de la rutina de CONSULTAS directas**

El diagrama de flujo de esta rutina se muestra en la Fig. 37.

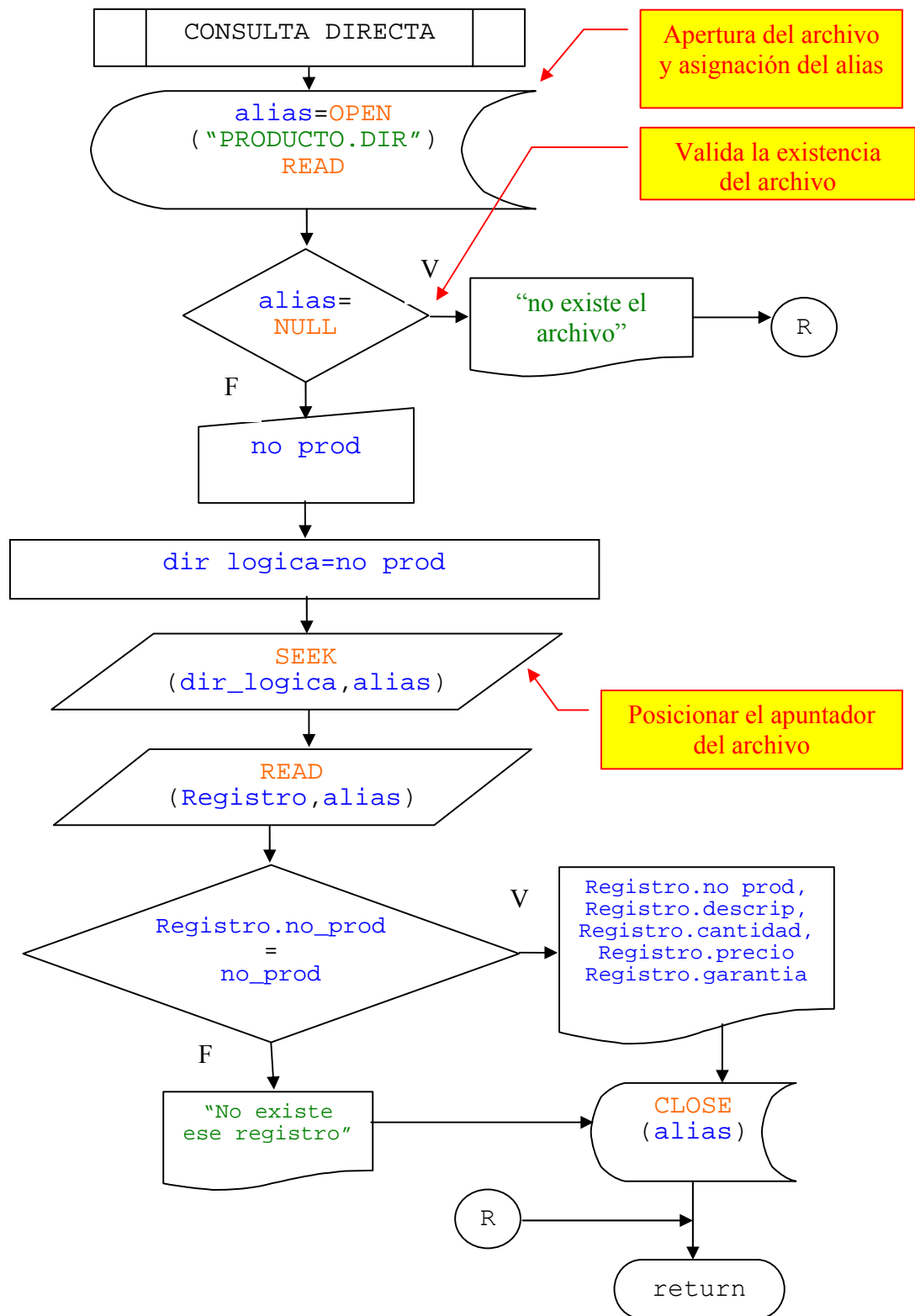


Fig. 37. Diagrama de flujo de rutina de consultas directas

### 3.3.2.2. Codificación de la rutina de CONSULTAS directas

La Fig. 38 muestra el código de la rutina de CONSULTAS directas de acuerdo al diagrama de la Fig. 37.

```

Procedure CONSULTA_DIRECTA;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
Begin
  Clrscr;

  writeln('CONSULTA DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.DIR'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.DIR
en modo de lectura/escritura}
{$I+}

  If(IoResult<>0) Then
  Begin
    writeln('No existe el archivo !!!');
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
    Exit;
  End;

  Repeat
    write('Numero de producto: '); Read(no_prod);
  Until no_prod<=FileSize(alias);
  Seek(alias,no_prod);
{$I-}
  Read(alias,Registro);
{$I+}
  { Lee el "Registro", de tamaño=sizeof(Registro) del archivo
"alias" }
  If(Registro.no_prod=no_prod) Then
  Begin
    writeln('No Prod          Descripcion  Cantidad
Precio  Garantia');
    writeln('-----');
    writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,'
',Registro.precio:4:2,' ',Registro.garantia);
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Close(alias);
    Readkey;
    Exit;
  End
  Else

```

```
writeln('No se encuentra ese registro !!!');  
Close(alias); { Cierra el archivo }  
writeln('<<< Oprima cualquier tecla para continuar >>>');  
Readkey;  
End;
```

*Fig. 38. Codificación de rutina de consultas directas*

### 3.3.3. MODIFICACIONES directas

Al igual que en el archivo secuencial, la forma de modificar el contenido de los campos de registros de un archivo, depende mucho de la rutina de consulta, ya que es necesario localizar previamente el registro que se desea modificar, capturar los nuevos valores y posteriormente **grabar el registro completo en la misma posición que se encontraba.** Esto último es muy importante porque recuerde que cuando se termina de leer un registro del archivo, el apuntador se posiciona al inicio del siguiente registro, por lo que, antes de grabar el registro modificado, es necesario reposicionar el apuntador del archivo en la dirección correcta.

#### 3.3.3.1. Diagrama de flujo de la rutina de MODIFICACIONES directas

La Fig. 39 muestra el diagrama de flujo de la rutina de modificaciones de campos de un registro particular en un archivo directo.

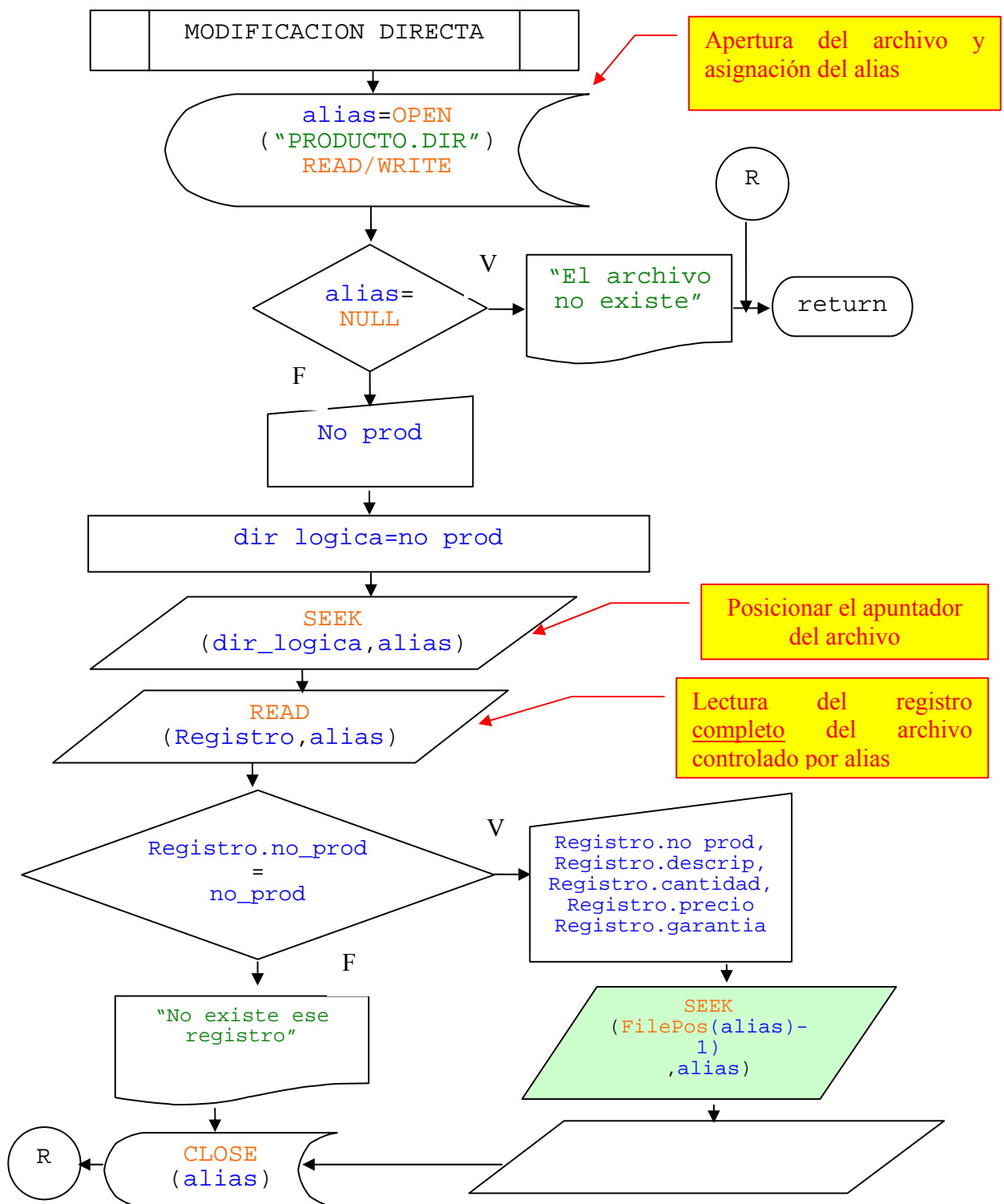


Fig. 39. Diagrama de flujo de rutina de modificación directa

### 3.3.3.2. Codificación de la rutina de MODIFICACIONES directas

La Fig. 40 muestra la codificación de la rutina de modificaciones de campos de un registro particular en un archivo directo.

```

Procedure MODIFICACION_DIRECTA;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
Begin
  Clrscr;

  writeln('MODIFICACION DE REGISTROS DE PRODUCTOS');

  Assign(alias,'PRODUCTO.DIR'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
  Reset(alias); {Intenta abrir el archivo PRODUCTO.DIR
en modo de lectura/escritura}
{$I+}

  If(IoResult<>0) Then
  Begin
    writeln('No existe el archivo !!!');
    writeln('<<<< Oprima cualquier tecla para continuar >>>>');
    Readkey;
    Exit;
  End;

  Repeat
    write('Numero de producto: '); Read(no_prod);
  Until no_prod<=FileSize(alias);
  Seek(alias,no_prod);
{$I-}
  Read(alias,Registro);
{$I+}
  { Lee el "Registro", de tamaño=sizeof(Registro) del archivo
"alias" }
  If(Registro.no_prod=no_prod) Then
  Begin
    writeln('No Prod          Descripcion  Cantidad
Precio  Garantia');
    writeln('-----');
    writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,'
',Registro.precio:4:2,' ',Registro.garantia);
    writeln('Anote los nuevos datos ...');
    write('Descripcion: '); Readln(Registro.descrip);
    write('Cantidad   : '); Readln(Registro.cantidad);
    write('Precio     : '); Readln(Registro.precio);
    Repeat
      write('Garantia   : '); readln(Registro.garantia);

```

```

        Registro.garantia:=Ucase(Registro.garantia);
        Until(Registro.garantia='S') OR (Registro.garantia='N');

    { Es necesario reposicionar el apuntador del archivo al principio
del
    registro que desea modificar, ya que al leer un registro, el
    apuntador se posiciona en el registro siguiente
    La funcion FilePos(alias) devuelve la posicion donde se encuentra
el
    apuntador }
    Seek(alias,FilePos(alias)-1);
    Write(alias,Registro); { Graba el registro con los nuevos campos}
    Close(alias); { Cierra el archivo }
    writeln('Registro modificado !!!');

    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
    Exit;
End
Else
    writeln('No se encuentra ese registro !!!');
    Close(alias); { Cierra el archivo }
    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
End;

```

*Fig. 40. Codificación de rutina de modificaciones directas*

### 3.3.4. Bajas de registros en un archivo de acceso directo (bajas lógicas)

Para el ejemplo que se ha estado manejando en este archivo directo, NO se pueden realizar bajas físicas (únicamente bajas lógicas), ya que existe una relación directa entre la dirección lógica y el número del producto: esto es, que el producto 1 se almacena en el registro lógico 1, o sea que cada producto se almacena en su respectivo registro de acuerdo al número de producto y si se aplicase la rutina de bajas físicas (compactación), los registros se recorrerían a otras direcciones que no corresponderían con su número de producto y dichos registros no podrían ser localizados por las rutinas de consultas, modificaciones y las mismas bajas lógicas.

Al igual que en las bajas lógicas de los archivos secuenciales, cuando se elimina un registro en forma lógica de un archivo, sólo se le coloca una marca



especial en alguno de sus campos que lo identifique como registro “borrado”, sin embargo, el registro sigue existiendo en el archivo y por lo tanto ocupa espacio.

### **3.3.4.1. Diagrama de flujo de la rutina de BAJAS lógicas directas**

Como se mencionó en el punto anterior, las bajas lógicas consisten en “marcar” los registros eliminados. En el ejemplo práctico que se muestra a continuación, el registro borrado se “limpia”, dejando en blanco todos sus campos (colocando el valor cero en los campos numéricos y blancos en las cadenas o de tipo carácter).

Para fines prácticos, la rutina de bajas lógicas se asemeja mucho a la rutina de modificaciones, sólo que en las bajas no se capturan los nuevos valores, sino se les asigna valores en blanco y se graba el registro completo en la misma posición en la que se encontraba. La diferencia entre las rutinas de bajas lógicas del archivo secuencial y el archivo directo radica en el mecanismo utilizado para localizar el registro que se desea eliminar. El diagrama de la Fig. 41 muestra esta rutina.

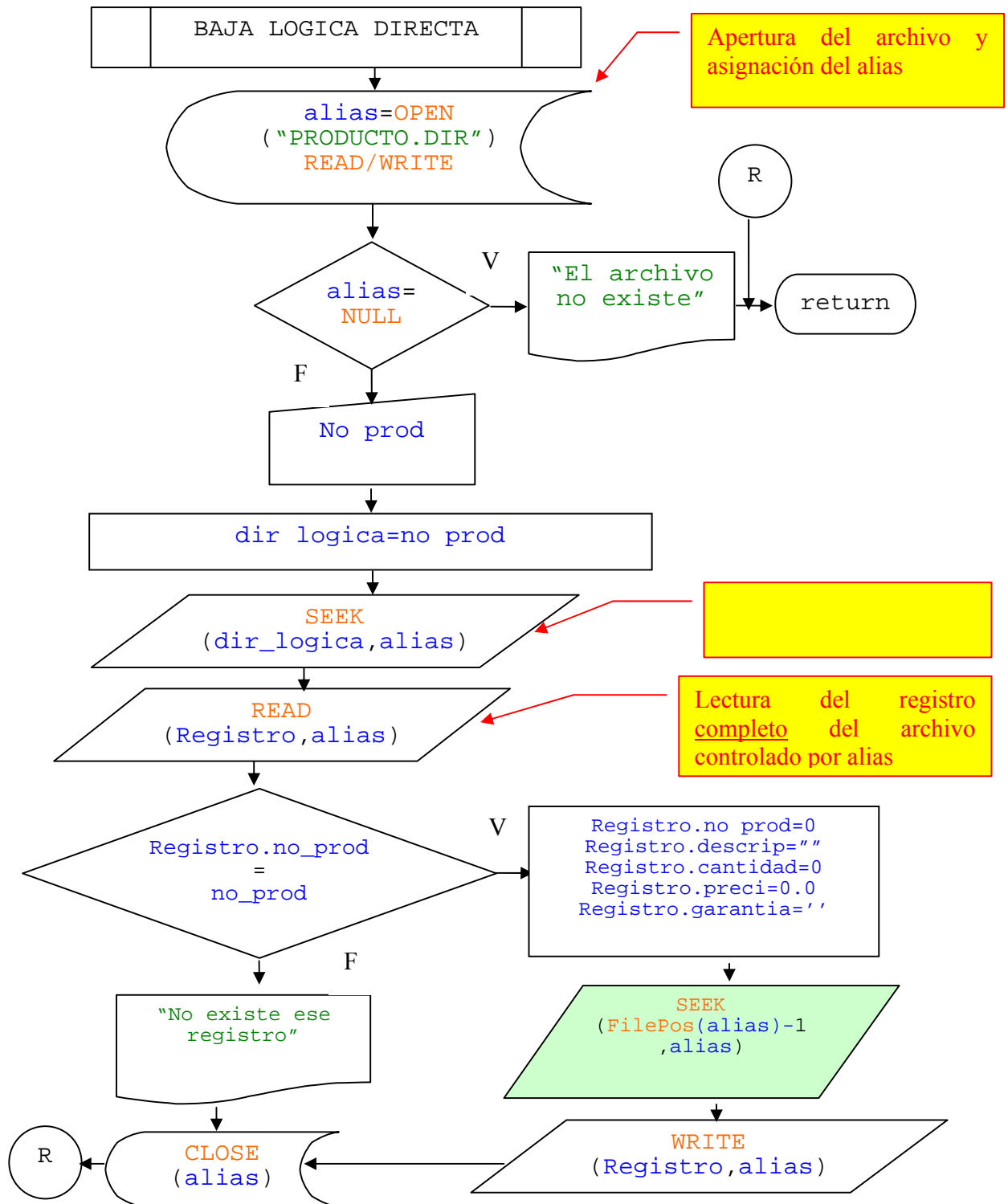


Fig. 41. Diagrama de flujo de rutina de baja lógica directa

### 3.3.4.2. Codificación de la rutina de BAJAS lógicas directas

La Fig. 42 muestra la codificación de la rutina de bajas lógicas en un archivo directo.

```

Procedure BAJA_LOGICA_DIRECTA;
Var no_prod : integer; { Variable local para el numero de producto que
desea consultar}
    op      : Char;
Begin
    Clrscr;

    writeln('ELIMINACION LOGICA DE REGISTROS DE PRODUCTOS');

    Assign(alias,'PRODUCTO.DIR'); { Asignacion del nombre del archivo al
alias}
{$I-} { <--- Directivas para deshabilitar mensajes de error del Sist.
Operativo }
    Reset(alias); {Intenta abrir el archivo PRODUCTO.DIR
en modo de lectura/escritura}
{$I+}

    If(IoResult<>0) Then
    Begin
        writeln('No existe el archivo !!!');
        writeln('<<< Oprima cualquier tecla para continuar >>>');
        Readkey;
        Exit;
    End;

    Repeat
        write('Numero de producto: '); Read(no_prod);
    Until no_prod<=FileSize(alias);
    Seek(alias,no_prod);
{$I-}
    Read(alias,Registro);
{$I+}
    { Lee el "Registro", de tamano=sizeof(Registro) del archivo
"alias" }
    If(Registro.no_prod=no_prod) Then
    Begin
        writeln('No Prod          Descripcion  Cantidad
Precio  Garantia');
        writeln('-----');
        writeln(Registro.no_prod:3,Registro.descrip:30,Registro.cantidad:3,'
',Registro.precio:4:2,' ',Registro.garantia);
        Registro.no_prod:=0;
        Registro.descrip:='';
        Registro.cantidad:=0;
    End;

```

```

        Registro.precio:=0.0;
        Registro.garantia:=' ';

    { Es necesario reposicionar el apuntador del archivo al principio
del
        registro que desea modificar, ya que al leer un registro, el
        apuntador se posiciona en el registro siguiente
        La funcion FilePos(alias) devuelve la posicion donde se encuentra
el
        apuntador }
    Repeat
        write('Esta seguro que desea borrarlo? [S/N] '); Readln(op);
        op:=Uppcase(op);
    Until (op='S') OR (op='N');

    If op='S' Then
    Begin
        Seek(alias,FilePos(alias)-1);
        Write(alias,Registro); { Graba el registro con los nuevos
campos}
        Close(alias); { Cierra el archivo }
        writeln('Registro eliminado !!!');
    End;

    writeln('<<< Oprima cualquier tecla para continuar >>>');
    Readkey;
    Exit;
End
Else
    writeln('No se encuentra ese registro !!!');
Close(alias); { Cierra el archivo }
writeln('<<< Oprima cualquier tecla para continuar >>>');
Readkey;
End;

```

*Fig. 42. Codificación de rutina de baja lógica directa*

## 4.CONCLUSIONES

Aunque existe una gran diversidad de aplicaciones que se pueden desarrollar con manejo de archivos que pueden ser sumamente completas y complejas, estos apuntes presentan, de una forma sencilla y comprensible, los aspectos básicos de programación de archivos usando Pascal. De tal forma, que no presenta lógica abrumadora de control de detalles, sino la base fundamental del material es entender y utilizar las funciones básicas de manejo de archivos en este lenguaje para posteriormente usarlas en el curso de “Administración de Archivos” y en cursos posteriores.

Cabe destacar que la codificación de las rutinas que se muestran se pueden obtener en forma íntegra en el sitio <http://www.itnuevolaredo.edu.mx/takeyas> y ejecutarse para reafirmar el contenido.

## 5. BIBLIOGRAFIA

- Dale, Nell. **"Pascal y estructuras de datos"**. McGraw Hill. USA
- Joyanes Aguilar, Luis. **"Problemas de Metodología de la Programación"**. McGraw Hill. 1990.
- Joyanes Aguilar, Luis. **"Programación en Turbo Pascal 7.0"**. Osborne.
- Loomis, Mary E.S. **"Estructura de Datos y Organización de Archivos"**. Prentice Hall. México. 1991.
- Martin, James. **"Organización de las bases de datos"**. Prentice Hall. 1993.
- Nyhoff, Larry R. **"Data Structures and Program Design in Pascal"**. Macmillan. USA
- O'brien, Stephen K. **"Turbo pascal 7.0. : Manual de referencia"**. McGraw-Hill . USA
- Rose, Cesar E. **"Archivos. Organización y Procedimientos"**. Computec. 1993.
- Sedgewick, Robert. **"Algorithms"**. Second edition. Addison Wesley. USA. 1988.
- Swan, Tom. **"Mastering Turbo Pascal 5.5"**. Third Edition. Hayden Books. USA.
- Tenenbaum Aaron M. **"Data Structures Using Pascal"**. Prentice-Hall . USA
- Tsai, Alice Y. H. **"Sistemas de bases de datos. Administración y uso"**. Prentice Hall. 1988.