

# Tema 1. Archivos.

## **Ejemplo: gestión de un archivo directo de productos**

Se desea gestionar un archivo de almacén. Se prevé que se va a tener un máximo de 100 productos distintos, y por cada uno de ellos se almacenará el código de producto, el código del proveedor, una descripción del producto y el stock.

Para un manejo más eficiente de los elementos se reservarán 120 posiciones de para almacenar los 100 productos. En cada una de esas 120 posiciones incluyen un campo `estado` de tipo entero que indica si la ranura está libre con un 0, ocupada con un 1, o dada de baja (baja lógica) con un 2 Por esta razón, se necesitará previamente ejecutar un archivo que inicialice las 120 posiciones y las marque como ocupadas.

```
algoritmo InicializarArchivoDirecto
const
    MaxReg = 120
tipos
    registro = rProducto
        entero : código
        cadena: desc
        entero : stock
        entero : estado
    fin_registro
    archivo_d de rProducto = aProducto
var
    aProducto : A
    rProducto : R
    entero : i
inicio
    crear ('PRODUCTOS.DAT')
    abrir (A, 'PRODUCTOS.DAT', escritura)
    R.estado ← 0 //Un 0 en el código indica que la ranura está vacía
    desde i ← 1 hasta MaxReg hacer
        escribir (A,R)
    fin_desde
    cerrar (A)
fin
```

## **Programa principal**

Para los algoritmos que aparecen a continuación, se suponen las siguientes declaraciones en el cuerpo del programa

```
algoritmo GestiónArchivoProductos
const
    MaxReg = 120
tipos
    registro = rProducto
        entero : código
        cadena: desc
        entero : stock
        entero : estado
    fin_registro
    archivo_d de rProducto = aProducto
var
    aProducto : A
inicio
    abrir (A, 'PRODUCTOS.DAT', lectura/escritura)
    //Cuerpo del programa principal
    cerrar (A)
fin
```

```
//Función de transformación de clave
entero función hash(valor rProducto : R)
inicio
    devolver(R.código mod MaxReg + 1)
fin_función

fin
```

## Mantenimiento del archivo

El mantenimiento incluirá los procedimientos para dar altas, bajas y modificaciones en al archivo

```
//El procedimiento Alta, pasa como argumento el registr (de tipo rProducto)
//que contendrá los datos del producto a dar de alta
procedimiento Alta(ref aProducto : A; valor rProducto : R)
var
    rProducto : RAux
    entero : NRR
inicio
    //La función buscar devuelve 0 si el registro no existe
    si buscar(A,R) = 0 entonces
        NRR ← hash(R)
        leer(A,NRR,RAux)
        //Si el registro tiene sinónimos, la posición ya está ocupada y el
        //campo estado es 1
        si RAux.estado = 1 entonces
            //Se busca, a partir de NRR una ranura libre para insertarlo
            repetir
                NRR ← NRR mod MaxReg + 1 //Para hacer un archivo circular
                leer(A,NRR,RAux)
            hasta_que (RAux.estado <> 1) //Hasta que una posición no esté ocupada
        fin_si
        //Si realmente sólo tenemos 100 productos, siempre habrá sitio
        //y siempre habrá que escribir
        escribir(A,NRR,R)
    si_no
        //El registro ya existe
    fin_si
fin_procedimiento
```

```
//La función Buscar, devuelve 0 si el código del producto
//del registro que se pasa como argumento no existe
//o el número de la ranura dónde está almacenado
entero : función Buscar(ref aProducto : A; valor rProducto : R)
var
    rProducto : RAux
    entero : NRR
inicio
    NRR ← hash(R)
    leer(A,NRR,RAux)
    //La ranura NRR está ocupada y es el registro buscado
    si RAux.codigo = 0 entonces
        devolver(0)
    si_no
        si (RAux.codigo = R.codigo) y (RAux.estado = 1) entonces
            devolver(NRR)
        si_no
            repetir
                NRR ← NRR mod MaxReg + 1
                leer(A,NRR,RAux)
            hasta_que (RAux.codigo = R.Codigo) y (RAux.estado = 1) o
                (RAux.estado = 0) //Hasta que se encuentra o
                //se llega a una ranura libre
            si (RAux.codigo = R.Codigo) y (RAux.estado = 1) entonces
```

```

        devolver(NRR)
    si_no
        devolver(0)
    fin_si
fin_si
fin_si
fin_procedimiento

```

```

//El procedimiento Baja borra un registro.
//El código del producto a borrar se pasa en el campo
//código del argumento rProducto
procedimiento Baja(ref aProducto : A; valor rProducto : R)
var

```

```

    rProducto : RAux
    entero : NRR,pos
inicio
    pos ← buscar(A,R)
    si pos <> 0 entonces
        //Un 2 en el campo estado, quiere decir que el registro no vale
        //pero que puede haber sinónimos más adelante
        R.estado ← 2
        escribir (A,pos,R)
    si_no
        //El registro no existe
    fin_si
fin_procedimiento

```

```

//El procedimiento Modificar cambia el contenido de un registro,
//teniendo en cuenta que el campo clave (el código de producto) no es modificable
//El argumento rProducto tiene el código del producto a modificar y el resto
//de datos ya modificados

```

```

procedimiento Modificar(ref aProducto : A; valor rProducto : R)
var
    entero : pos
inicio
    pos ← buscar(A,R)
    si pos <> 0 entonces
        escribir (A,pos,R)
    si_no
        //El registro no existe
    fin_si
fin_procedimiento

```