Actualización de Archivos Binarios

Redictado FOD

A continuación, se define la interfaz (tipos y firmas de subprogramas públicos) de una unidad en Free Pascal como protocolo para la manipulación de un archivo binario de registros de longitud fija como tipo de dato abstracto.

Se define la unidad para un archivo binario con mantenimiento de espacio libre en forma de pila (LIFO: Last In First Out) a partir del registro 0 del archivo. Los registros se definen como variantes, dependiendo del campo booleano valido, que indica si contiene un dato válido o es un registro de encadenamiento de posiciones libres. Los registros definidos de esta forma tienen un tamaño en bytes que corresponde a la variante de mayor tamaño, pero la estructura y acceso a los campos depende del valor del campo que define a la variante (no puede haber campos homónimos en secciones variantes distintas).

El tipo para los registros de datos se define empaquetado *(packed)* porque Pascal utiliza por defecto una cantidad par de bytes para almacenar cada campo de un registro; por tanto, si en un registro hay campos cuyos tipos tienen valores que se representan en una cantidad impar de bytes, el programa le agregará un byte extra que quedará inutilizado. Esto último es ineficiente cuando los registros se almacenan en un medio de almacenamiento masivo, porque implica una importante cantidad de espacio desperdiciado en casos de archivos voluminosos. El empaquetamiento fuerza a que el programa almacene los campos de un registro en forma adyacente sin entornarlos a una cantidad par de bytes.

Se define un registro de control del archivo para agrupar todas las variables necesarias para poder manipularlo y minimizar el acoplamiento de subprogramas que operen con éste.

**Unit** tProducto;

**Interface**

**Type**

tReg = **Packed Record** *{tipo de registro para producto}*

**Case** valido: **Boolean** **of** {discriminante de registros para comprobar si son válidos (de producto) o no (de encadenamiento de posiciones libres; en el registro empaquetado ocupa 1 byte, si no, ocuparía 2)}

*// Variante de registro de datos válido:*

**True**: ( id: **Longword**;

descrip: **String**[30]; *{en el registro empaquetado ocupa 31 bytes (por el byte inicial de longitud), si no, ocuparía 32}*

precio: **Real**;

stock: **Word** );

*// Variante de registro de datos inválido o de encadenamiento de posiciones libres:*

**False**: ( proxLibre: **Longword** ); {si proxLibre=0, es la última posición libre en el archivo o el registro en la posición 0 indicando que no hay espacio libre en el archivo}

**end**;

tArch = **File of** tReg; {tipo de archivo para registros de producto con registro de control de espacio libre en la posición 0}

tCtlArch = **Record** {tipo de registro de control para archivo de productos}

nomarch: String; {nombre físico del archivo}

arch: tArch;

r: tReg; {registro buffer para lectura o escritura de productos}

libre: Longword {posición del primer registro libre (para ahorrar la lectura del primer registro del archivo cuando se requiera agregar un nuevo registro de producto); se inicializa al crear o abrir el archivo}

**end**;

tResult = (ok, duplicado, inexistente); {resultados posibles}

//Primitivas de creación y control general:

**Procedure** Crear (**var** a: tCtlArch); {recibe en a.nomarch el nombre a asignar a a.arch, crea el archivo con el registro 0 indicando que no hay espacio libre, y lo cierra}

**Procedure** Abrir (**var** a: tCtlArch); {recibe en a.nomarch el nombre a asignar a a.arch, abre el archivo, lee el registro 0 en a.r e inicializa a.libre con r.proxLibre}

**Procedure** Cerrar (**var** a: tCtlArch); {sólo cierra a.arch, ya que, por convención, el registro 0 siempre se mantiene actualizado}

//Primitivas de recuperación:

**Procedure** Leer (**var** a: tCtlArch; **var** cod: tResult); {recibe a.arch abierto para leer y escribir y devuelve en a.r al siguiente registro válido del archivo a partir de la posición actual y cod=ok, o cod=inexistente si no hay siguiente}

**Procedure** Buscar (**var** a: tCtlArch; idp: **Longword**; **var** cod: tResult); {recibe a.arch abierto para leer y escribir y devuelve en a.r al registro del archivo con a.r.id=idp y cod=ok si lo encuentra, o cod=inexistente si no}

**Procedure** Reportar (**var** a: tCtlArch; **var** reporte: Text); {recibe a.arch abierto para leer y escribir y reporte asignado y sin abrir. Recorre todo el archivo secuencialmente reportando sus registros válidos en el archivo de texto, uno por línea, con sus campos en el mismo orden que en tReg y encolumnados}

{Buscar y Reportar deben codificarse empleando la primitiva Leer}

//Primitivas de actualización

**Procedure** Insertar (**var** a: tCtlArch; **var** rp: tReg; **var** cod: tResult); {recibe a.arch abierto para leer y escribir y devuelve cod=ok si lo pudo insertar, o cod=duplicado si encontró un registro con a.r.id=rp.id. La inserción debe efectuarse en el primer espacio libre, si hubiera, o al final del archivo, si no}

**Procedure** Eliminar (**var** a: tCtlArch; idp: **Longword**; **var** cod: tResult); {recibe a.arch abierto para leer y escribir y devuelve cod=ok si se pudo eliminar, o cod=inexistente si no. Actualiza el primer registro libre}

**Procedure** Modificar (**var** a: tCtlArch; **var** rp: tReg; **var** cod: tResult); {recibe a.arch abierto para leer y escribir y al registro modificado en rp, y devuelve cod=ok si lo pudo modificar, o cod=inexistente si no se encontró un registro con a.r.id=rp.id}

{Para buscar registros para control de unicidad en Insertar, para Eliminar y para Modificar, se debe usar la primitiva Buscar}