Programación Modular

Examen Ord. ETSI Telecomunicación 23 de junio, 2005

APELLIDOS, Nombre | Grupo | NºConv

Entregar esta hoja. *Poner "Apellidos, Nombre", letra del Grupo y número de la convocatoria.*

1 Responder [2]

a) La recursividad es más ineficiente que iteración. Indicar un motivo técnico y un motivo algorítmico de tal ineficiencia y, por último, el motivo técnico de los posibles desbordamientos de memoria en los problemas recursivos. Ser breves.

b) Suponiendo un árbol de búsqueda simple dibujar los árboles que se van formando tras añadirle: $10, 5, 12, 13, 7, 8, 3, 11$ y asímismo cada árbol que va quedando tras borrarle: $10, 5, 12, 11$

c) Suponer que tenemos un array de punteros a cadenas de caracteres terminadas en el caracter nulo. El último puntero es también el puntero nulo. Hacer un segmento de programa que escriba en la pantalla el contenido de todas estas cadenas.

d) Suponer un puntero a un array de registros que contiene un campo nombre, apellidos y edad. Definir los tipos y mostrar como se accedería a la primera letra del nombre de la persona que está en la posición tercera del array de registros. Indicar la operación.

2 Dada una estructura en forma de lista (no TAD) de nodos simplemente enlazados, hacer un procedimiento de purga, que reciba tal lista y elimine los repetidos pero contándolos. Para ello, suponer que los nodos además de la información de tipo base (`TBase`) contienen un contador (Natural). Este contador está inicialmente a 1 cuando se manda la lista al procedimiento de purga. El procedimiento de purga modificará la lista eliminando los nodos repetidos y dejará en el contador la candidad de nodos repetidos que había de cada tipo. Por ejemplo: $(B,1),(A,1),(A,1),(B,1),(M,1)$ quedaría: $(B,2),(A,2),(M,1)$ [3]

3 Supónganse las clases `Fichero` y `Cadena` cuyos respectivos interfaces son: [2.5]

```
INTERFAZ CLASE FICHERO
MÉTODOS
  Abrir(E CADENA nomfich) // abre el fichero de nombre nomfich.
                          // Para leer de un fichero es necesario
                          // que esté abierto.
  LeerLinea(S CADENA ln)  // lee una linea del fichero y la guarda
                          // en ln. La siguiente lectura devuelve
                          // la siguiente línea
  B FinFichero()          // devuelve verdadero si la última
                          // lectura no fue posible debido
                          // a haberse alcanzado el final del fichero
  Cerrar()                // Cierra el fichero. Tras cerrar
                          // un fichero ya no se puede leer más de él.
FIN

INTERFAZ CLASE CADENA
MÉTODOS
  N Longitud()                  // devuelve el número de caracteres de la cadena
  C ConsultarPos(E N pos)       // devuelve el carácter que está en la
                                // posición pos de la cadena. El rango válido
                                // de posiciones es entre 1 y la longitud.
  InsertarPos(E C car, E N pos) // Inserta un carácter en la posición pos de
                                // la cadena. La longitud de la cadena aumenta
                                // en 1 y no se sobreescribe ningún valor. El
                                // rango válido de posiciones es entre 1 y la
                                // longitud + 1.
  BorrarPos(E N pos)            // Borra el elemento de la posición pos. La
                                // longitud de la lista disminuye en un
                                // elemento. El rango válido de posiciones es
                                // entre 1 y la longitud.
FIN
```

Codificar un algoritmo que haga uso de esas dos clases, tome como parámetro una cadena que representa el nombre de un fichero y devuelva una cadena que contenga las letras que están en todas las líneas del fichero.

4 Crear las siguientes operaciones sobre árboles binarios: 2.5

a) `B ALGORITMO Iguales(E NodoArbol *a, *b)`, que devuelve `VERDADERO` si ambos árboles tienen la misma forma y los mismos valores en los nodos que estén en la misma posición en ambos árboles.

b) `B ALGORITMO Subárbol(E NodoArbol *a, *b)`. Esta operación devuelve `VERDADERO` si el segundo árbol es un subárbol del primero y `FALSO` en caso contrario. Un árbol es un subárbol de otro cuando es igual (según la definición de la primera operación) que alguno de los hijos de cualquiera de los nodos de un árbol

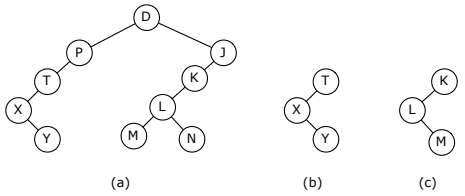

Figura 1: El árbol (b) es subárbol de (a), mientras que (c) no lo es.

Los árboles se han de tratar a nivel de implementación. La definición del tipo `NodoArbol` es:

```
REGISTRO NodoArbol
  TElem dato
  NodoArbol *izq, *der
FIN
```