Examen Final de PRED (Enero 2007)

1. **(8 puntos)** Para un programa de análisis de textos se desea implementar una estructura que nos diga cuantas veces determinados pares de palabras aparecen en un texto dentro de la misma frase. En concreto, se desea que esta estructura tenga las siguientes operaciones:

   - crear: nos crea la estructura vacía.
   - incrementar: dadas dos palabras P1 y P2, se incrementa el número de veces que la pareja de palabras <P1,P2> ha aparecido.
   - consulta1: dadas dos palabras P1 y P2 nos devuelve el número de veces que han aparecido juntas esas dos palabras en cualquier orden. En particular, esto quiere decir que, dada una estructura E, si se han realizado las operaciones E.crear; E.incrementar("mi", "casa"); E.incrementar("mi", "casa"); E.incrementar("tu", "casa"); E.incrementar("casa", "mi");y a continuación se hace la operación E.consulta1("mi", "casa"); la respuesta debería de ser 3, igual que si se realiza la operación E.consulta1("casa", "mi"). Análogamente, si hiciéramos las consultas E.consulta1("tu", "casa"); o E.consulta1("casa", "tu"); la respuesta debería de ser 1.
   - listar: nos escribe la lista, por orden alfabético creciente, de todas las palabras que están en la estructura de datos. Por ejemplo, si se realizaran las operaciones descritas más arriba, la operación E.listar debería escribir la lista:

     casa
     mi
     tu

   Para implementar esta estructura de datos se ha decidido utilizar, por una parte, una lista encadenada de palabras, ordenada por orden alfabético, que incluya todas las palabras que se han añadido a la estructura de datos y, por otra, una tabla de hash encadenado que, a cada par de palabras, P1 y P2, nos devuelva el número de veces que han aparecido (es decir la respuesta a la operación de consulta1(P1.P2)). Es decir:

```
clase Estructura
     primero_lista: ^nodo1;
     t_hash: tabla [0..N] de ^nodo2;
     ....
     /* operaciones de la clase */
     ...
f_clase
clase nodo1
     palabra: string;
     sig: ^nodo1;
f_clase
```

```
clase nodo2
     palabra1: ^nodo1;
     palabra2: ^nodo1;
     cont: entero;
     sig: ^nodo2;
f_clase
/* para ahorrar memoria los campos palabra1 y palabra2 son
punteros a la lista de palabras y no strings: en vez de
guardar una misma palabra (que podría ser relativamente
larga: por ejemplo hasta 15 letras) varias veces la
guardamos una única vez en la lista de palabras, mientras
que en la tabla sólo guardamos punteros a esa palabra */
```

Se pide:

a) Especificar la implementación, dando el invariante y la equivalencia de la representación.

b) Supongamos que tenemos ya implementada una función de hashing h1 que, dado un string, nos devuelve un valor entre 0 y N. Proponed una implementación de la función de hashing h2 (utilizando la función h1) que es necesaria para implementar la estructura de datos del problema (h2 debe de devolver un valor entre 0 y N para cada par de palabras).

c) Implementar la operación incrementar, justificando informalmente su corrección.

d) Supongamos que se desea tener otra forma de consulta, que queremos que sea muy eficiente. En concreto una operación consulta2 que dado un número n nos diga cuantas palabras de n letras hay incluidas en la estructura de datos. Indicad que modificaciones sería conveniente realizar en la implementación de la estructura de datos para hacer esta consulta eficiente.

2. **(2 puntos)** Un "heap" (además de una técnica de implementación de la gestión dinámica de memoria) es un tipo de árbol binario que tiene la propiedad de que el valor que hay en cada nodo es menor que los valores que hay en sus dos hijos. La forma de implementar un heap consiste símplemente en utilizar una tabla t: **tabla** [1..N] **de** valores (por ejemplo, enteros si los valores guardados en el heap son números enteros) de tal manera que la raiz se supone que está en la posición 1 de la tabla; sus hijos están en la posición 2 y 3; los hijos del nodo t[2] estan en t[4] y t[5]; los de t[3] estan en t[6] y t[7]; y, en general, los hijos del nodo en la posición t[i] estan en t[2*i] y t[2*i+1], si $2*i+1 \leq N$. Esto quiere decir, en particular, que todas las ramas del heap tienen más o menos la misma altura.

   Se pide diseñar un algoritmo que, dada una tabla t: **tabla** [1..N] **de** enteros que se supone que implementa un heap, recorra el árbol en preorden y nos diga si cumple la propiedad del heap, es decir si el valor en cada nodo es menor que el valor de sus hijos.