CENTRO SUPERIOR DE INFORMÁTICA
Departamento de Estadística, I.O. y Computación

Teoría de Autómatas y Lenguajes Formales
Curso 2002-2003

PRACTICA 3: Tablas Hash II

Diseñar e implementar un programa en C++ que lea todas las palabras de un fichero de texto y las incluya en una tabla hash. Se considerará una palabra cualquier secuencia de caracteres delimitados por espacios en blanco, y se eliminarán de la entrada todos los signos de puntuación (puntos, comas, puntos y comas, dos puntos, etc.)

El tamaño de la tabla hash vendrá dado por un valor `SIZE_TABLE`, que debería ser un número primo.

La función hash a utilizar será del tipo módulo: a cada palabra se le asociará un número, y el valor hash para la palabra será el resto de dividir dicho número por el tamaño de la tabla. Para asociar el número a la palabra, se sumarán los valores numéricos (códigos ASCII) correspondientes a cada uno de sus caracteres, ponderando estas sumas (se puede usar el mismo tipo de función que en la práctica anterior).

La tabla consistirá en un vector, cada uno de cuyos elementos será un puntero a una lista enlazada. En este caso las colisiones se resolverán mediante *encadenamiento directo*: cuando a dos palabras les corresponda la misma posición en la tabla hash, las palabras deberán aparecer encadenadas en la lista apuntada por el puntero correspondiente a esa posición, tal como muestra la figura 3.1.

La figura 3.1 muestra el contenido del fichero `hash.h` en el que se definen la clases `HashTable` y `list`.

La clase `HashTable` contiene 2 atributos privados: el tamaño de la tabla y un puntero a listas, llamado `tabla`. Este puntero se utilizará para construir, de forma dinámica un vector de listas, es decir, un vector donde cada uno de sus elementos será un objeto lista.

Los métodos que suministra esta clase son el constructor, el destructor y métodos para insertar, buscar y eliminar una palabra de la tabla.

El método `hash` es el encargado de calcular la posición que le corresponde en la tabla a la cadena que se le pasa como parámetro.

Obsérvese que algunos de los métodos de la clase `HashTable` (por ejemplo `find`, definido en la línea 30) llevan el cualificador `const`. Recuerde que cuando un método está cualificado como `const` el método no puede modificar

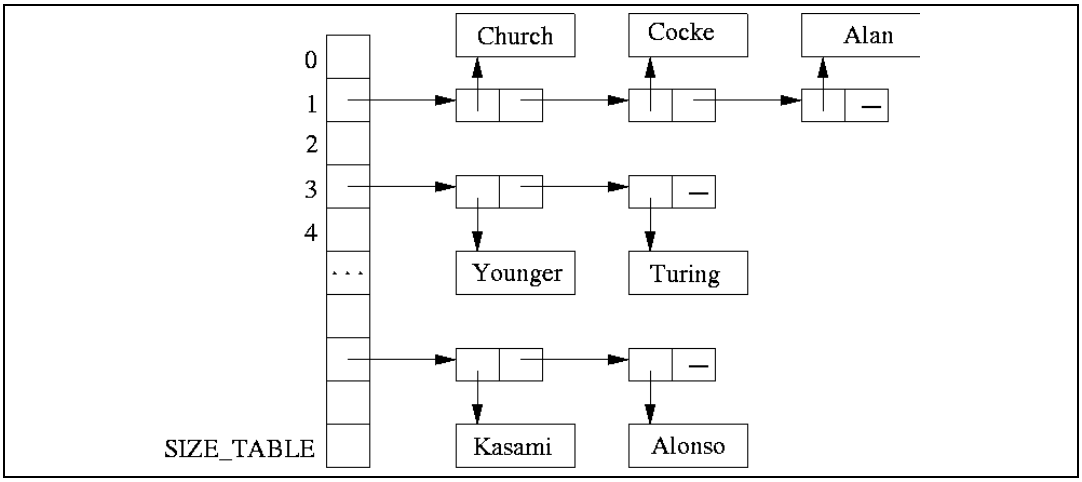

Figura 3.1: Estructura de la tabla hash

ninguno de los atributos de la clase. Por otra parte, un método cualificado `const` puede ser invocado por objetos constantes y no constantes.

El fichero `hash.h` también contiene las definiciones correspondientes a la clase `list`, que se utiliza para gestionar las listas de elementos a las que apuntan las casillas de la tabla hash.

El único atributo de una lista es un puntero `pStart` a elementos siendo los elementos la estructura que se define en la línea 4, que consta de un puntero a caracteres (para almacenar la cadena), un campo numérico `num` en el que se puede almacenar cualquier valor numérico asociado con la cadena, por ejemplo el número de línea en que la cadena aparece en el texto, y un puntero al siguiente nodo de la lista.

Para la implementación de esta práctica lo/as alumno/as deberán codificar el contenido del fichero `hash.C`. Este fichero deberá contener la implementación de todos los métodos definidos en `hash.h`.

Para probar el programa debe diseñarse también un programa cliente que:

1. Leerá desde la línea de comandos un nombre de fichero

2. Si en la línea de comandos no se suministra un nombre de fichero, se imprimirá un mensaje de error y se finalizará la ejecución

3. Se abrirá el fichero, y se leerán todas las palabras que contenga, insertándolas en la tabla hash

4. Después el programa entrará en un bucle en el que pedirá al usuario una palabra y le indicará si la palabra estaba o no en el fichero, buscándola en la tabla hash

5. El programa finalizará cuando el usuario introduzca la palabra *FIN* escrita en mayúsculas

```
1  #include <iostream.h>
2  #include <stdlib.h>
3
4  struct elem{char *name; unsigned num; elem *next;};
5
6  class list {
7  public:
8     list();
9     ~list();
10    void RemoveNode(const char *s);
11    void ListInsert(const char *s, unsigned num);
12    elem *FindPosition(const char *s)const;
13    void remove(elem *p);
14    list(const list& s) {
15      cout << "La copia no está permitida." << endl; exit
    (1);
16    }
17    list &operator=(const list &s) {
18      cout << "La asignación no está permitida." << endl
    ; exit(1);
19      return *this;
20    }
21 private:
22    elem *pStart;
23 };
24
25 class HashTable {
26 public:
27    HashTable(unsigned len=1021);
28    ~HashTable();
29    void insert(const char *s, unsigned num);
30    elem *find(const char *s) const;
31    void remove(const char *s);
32    unsigned hash(const char *s) const;
33 private:
34    unsigned size_table;
35    list *tabla;
36 };
```

Figura 3.1: El fichero hash.h

Enumeramos a continuación los pasos que habitualmente se siguen cuando se utiliza una metodología orientada a objetos en el diseño de un programa:

1. En el ámbito de aplicación de su programa, identifique las entidades (objetos) y sus atributos (datos).

2. Determine las acciones que pueden realizarse sobre un objeto.

3. Determine las acciones que un objeto puede realizar sobre otros objetos.

4. Determine las partes de cada objeto que serán visibles a otros objetos, qué partes serán públicas y cuáles privadas.

5. Defina la interface pública de cada objeto.

Como mejora opcional para la implementación de esta práctica, l@s alumn@s podrían estudiar los aspectos comunes que encuentren entre ésta implementación y la que realizaron en la práctica anterior, y utilizando programación genérica tratar de diseñar una clase que soporte tablas hash en las que se pueda almacenar diferentes tipos de estructuras de datos.

Otra posibilidad es implementar, también utilizando programación genérica una clase lista en la que se pueda almacenar diferentes tipos de elementos.