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

Teoría de Autómatas y Lenguajes Formales
Curso 2002-2003

PRACTICA 2: Tablas Hash

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 separadores. Se considerarán separadores los siguientes caracteres: espacio(' '), nueva línea ('\n'), tabulador ('\t'), salto de página ('\f'), retorno de carro ('\r'), coma (','), punto ('.'), punto y coma (';') o dos puntos (':').

Los separadores no pueden formar parte de las palabras, y al procesar el fichero de texto, deberán ignorarse. Estudiar la información (man) correspondiente a las funciones `isalpha()`, `isnum()` y similares de C++, por si fuera interesante utilizarlas. Asimismo recomendamos buscar información sobre las funciones de C++ contenidas en la librería `string.h`. Más concretamente, pueden resultar de interés las funciones `strcpy` y `strcmp`.

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 (véase el código de la función `hash` en la figura 2.2).

La tabla consistirá en un vector, cada uno de cuyos elementos será una estructura (`struct`) que contendrá dos campos: una cadena de caracteres de un cierto tamaño, y un número asociado con la cadena:

```
struct elem {char word[LEN_CADENAS]; unsigned num;};
```

En la cadena se almacenarán los caracteres de la cadena que se insertan en la tabla, y en el campo numérico se puede almacenar cualquier valor numérico asociado con la cadena, por ejemplo el número de línea en que la cadena aparece en el texto.

La resolución de colisiones se llevará a cabo mediante *direccionamiento abierto*: si el slot en el que se pretende insertar una palabra resulta que ya está ocupado, la estrategia que se sigue es buscar el siguiente slot que esté

libre (teniendo en cuenta no desbordar la tabla). Para usar este método será necesario inicializar de alguna manera todas las posiciones de la tabla. Por ejemplo, se puede colocar inicialmente la cadena vacía ("") en todas las posiciones.

```
#include <iostream.h>

const unsigned LEN_CADENAS = 30;
const unsigned SIZE_TABLE = 1021;
struct elem {char word[LEN_CADENAS]; unsigned num;};

class HashTable {
public:
   HashTable(unsigned len = SIZE_TABLE);
   ~HashTable(){delete[] a;}
   void insert(const char *s, unsigned num);
   elem *find(const char *s);
private:
   unsigned hash(const char *s);
   unsigned size_table;
   elem *a;
};
```

Figura 2.1: El fichero hash2.h

La figura 2.1 muestra el contenido del fichero `hash2.h` en el que se define la clase `HashTable`.

La clase contiene 2 atributos privados: el tamaño de la tabla (`SIZE_TABLE`), y un vector de elementos (realmente es un puntero a elemento, que correspondería con el primero de los elementos del vector), `*a`.

Como métodos públicos la clase tiene un constructor, un destructor y dos métodos para inserción y borrado.

El método privado `hash` es el que calcula el valor hash correspondiente a una determinada cadena, y su código aparece en la figura 2.2, que es parte del fichero `hash.C`.

```
unsigned HashTable::hash(const char *s) {
   unsigned sum = 0;
   for (unsigned i = 0; s[i] != '\0'; i++)
     sum += (i + 1) * unsigned(s[i]);
   return sum % size_table;
}
```

Figura 2.2: El método `hash`

Para la implementación de esta práctica lo/as alumno/as deberán codificar el contenido del fichero `hash.C`. Este fichero deberá contener la implementación del resto de los métodos definidos en `hash2.h`, es decir:

```
HashTable(unsigned len=1021);
void insert(const char *s, unsigned num);
elem *find(const char *s);
```

El método `find` deberá devolver un puntero a la tabla de símbolos que apuntará al elemento de la tabla que contiene la palabra buscada o bien un puntero `NULL` en caso de que no encuentre la palabra en la tabla.

El código que se presenta en la figura 2.3 es un ejemplo de programa que utiliza la clase HashTable. Lee cadenas de un fichero de entrada, y sin procesarlas de ninguna manera, las inserta en la tabla hash. Luego solicita al usuario que introduzca cadenas y le indica si se encuentran o no en la tabla.

```
// hash2ap: Hashing con direccionamiento abierto
//     (Es necesario enlazar con hash2.C)
//     F. de Sande
//     Prácticas de TALF 11.Oct.2002
//
#include <fstream>
#include <stdlib.h>
#include <iomanip>
#include <string>
#include "hash2.h"

const unsigned SIZE = 631;

int main(int argc, char *argv) {
   ifstream Fichero("hash.txt", ios::in);
   if (!Fichero) {
      cout << "Imposible abrir hash.txt.\n";
      exit(1);
   }
   string cadena;
   elem *p = NULL;

   HashTable Tabla(SIZE);
   cout << "Datos leídos del fichero hash.txt\n" <<
      endl << endl;
   while (Fichero >> cadena) {
     cout << "Cadena: " << cadena << endl;
```

```
      Tabla.insert(cadena.c_str(), 5);
    }

    cout << endl;

    for (;;) {
       cout << "Introduzca una palabra o ! para salir: "
          ;
       cin >> cadena;
       if (!cadena.compare("!"))
         break;  // Salimos del bucle
       p = Tabla.find(cadena.c_str());
       if (p)
          cout << "La palabra SÍ estaba en el fichero "
              << endl;
       else cout << "No la he encontrado." << endl;
    }

    return 0;
}
```

Figura 2.3: Un programa que utiliza la clase HashTable

En el caso del programa que se debe desarrollar, antes de insertar las cadenas en la tabla hash, habrá que eliminar los separadores de las cadenas.

Para más información sobre la técnica de hashing, podemos recomendar la obra ya clásica *Algoritmos + Estructuras de Datos = Programas* de N. Wirth.