Guion de prácticas 5

Punteros y Memoria Dinámica

DECSAI

Metodología de la Programación

Grado en Ingeniería Informática

Prof. David A. Pelta

Introducción al guion

En este guion se pondrán en práctica los conceptos de punteros y memoria dinámica. Repase los apuntes de teoría.

Diccionario con índice

Un sistema de reconocimiento de voz dispone de un diccionario o catálogo de palabras reconocidas. Este diccionario se puede representar mediante un array de strings, al que llamaremos `datos`, donde cada elemento es una palabra que el sistema puede reconocer.

Para simplificar la gestión del diccionario, se quiere construir un índice como el que se muestra en la Fig. 1 (a). Dicho índice se puede representar como un array de punteros a datos de tipo strings que llamaremos `indice`. Así, `indice[0]` apunta a la primera palabra en `datos` que empieza con 'A', `indice[1]` a la que empieza con 'B' y así sucesivamente. En la posición `indice['Z'-'A' + 1]` se almacena un puntero al final de los elementos utiles del array `datos`. Note que el array `indice` tiene $n\_indice =$ `'Z'-'A' + 2` componentes.

Se pretende desarrollar un módulo `Diccionario` para gestionar algunas operaciones básicas sobre el diccionario y el índice.

Estas funciones (decida que parámetros son necesarios) son:

- `crearDiccionario`: inicializa el diccionario y el índice. En el array `datos` se almacenarán los valores "A", "B", "C" y así sucesivamente. El resultado puede verse en la Fig. (b). Para el array `datos` se reservarán $n\_datos = n\_indice \times 4$ componentes. Debe utilizar un contador n_utiles para las posiciones utilizadas.
- `mostrarPalabras`: recibe el array `indice` y una letra $inicial$ y muestra todas las palabras del diccionario que comienzan con $inicial$.
- `mostrarDiccionario`: utilizando la función anterior, muestra todas las palabras del diccionario.
- `cuentaPalabras`: devuelve la cantidad de palabras que empiezan con una letra dada. NO DEBE RECIBIR EL ARRAY `datos`.
- `existe`: comprueba si una palabra existe en el diccionario. Devuelve `true` o `false`.
- `agregaPalabra`: si hay espacio en `datos`, agrega la palabra `pal` si no existe en el diccionario. Idealmente, la palabra debe insertarse de forma ordenada en `datos`. Caso contrario, puede agregarla al final de las palabras que comienzan con la letra `pal[0]` (el primer carácter de `pal`). Cada inserción produce un desplazamiento de valores en el array `datos`. Por tanto, es necesario actualizar también el `indice`.

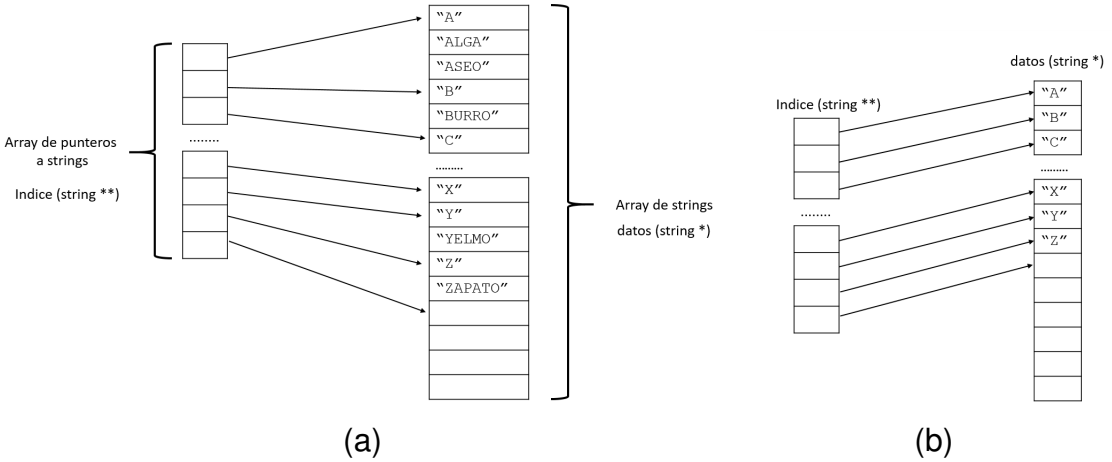

Figura 1: Representación del diccionario e índice (a). En (b), estado inicial.

- `borraPalabra`: elimina la palabra `pal` si existe en el diccionario. Cada borrado produce un desplazamiento de valores en el array `datos`. Por tanto, es necesario actualizar también el `indice`.

Puede suponer que todas las palabras están en mayúsculas.

Las inserciones/borrados se realizan haciendo desplazamientos a derecha/izquierda. Recuerde actualizar el valor n_utiles.

Tareas a Realizar

1. Describa la salida del código mostrado en la Fig. 3. Escriba, compile y ejecute el código para comprobar si lo ha entendido correctamente.

2. A partir de un directorio raíz `practica5`, cree un proyecto de NetBeans llamado `diccionario` (con las correspondientes carpetas `src, include`).

   Implemente el módulo descrito y escriba un fichero `main.cpp` donde pruebe todas las funciones. Se sugiere un esquema como el mostrado en la Fig. 2

```
Reservar memoria
Crear diccionario
Agregar palabras desde un fichero (redirección de entrada)
Comprobar si existe una palabra (probar ambos casos)
Para cada letra entre ‘‘A’’ y ‘‘Z’’
        mostrar cuantas palabras existen

Mostrar el diccionario completo
Mostrar cuantas palabras existen con ‘E’
Borrar una palabra que empiece con ‘E’
Mostrar cuantas palabras existen con ‘E’
Liberar memoria
```

Figura 2: Estructura sugerida para probar el módulo

```
void mostrar(int *v, int k){
  for(int i = 0; i < k; i++)
     cout << v[i] << ", ";
  cout << endl;
}

int main()
{ const int N = 10;
  int *datos;
  datos = new int[N];

  int signo = 1;
  for(int i = 0; i < N; i++){
    datos[i] = i * signo;
    signo = signo * -1;
   }

  mostrar(datos, N);

  int *p = &datos[2];
  mostrar(p, 5);

  p = &datos[4];
  mostrar(p, 5);

  int *p1, *p2;
  p1 = &datos[3];
  p2 = &datos[8];
  cout << "Salida 1: " << p2 - p1 << endl;

  p1 = &datos[0];
  cout << "Salida 2: " << p2 - p1 << endl;

  p1 = datos;
  cout << "Salida 3: " << p2 - p1 << endl;

  p1 = &datos[5];
  cout << "Salida 4: " << *p2 << " - " << *p1
       << " = " << *p2 - *p1 << endl;

  delete [] datos;
}
```

Figura 3: Describe la salida del código