UNIVERSIDAD DE MÁLAGA

Departamento de Lenguajes y Ciencias de la Computación

LABORATORIO DE PROGRAMACIÓN

Examen Ordinario de Junio

1º Gestión Curso 2009/2010

23/06/2010 a las 16:30

APELLIDOS _____ NOMBRE _____

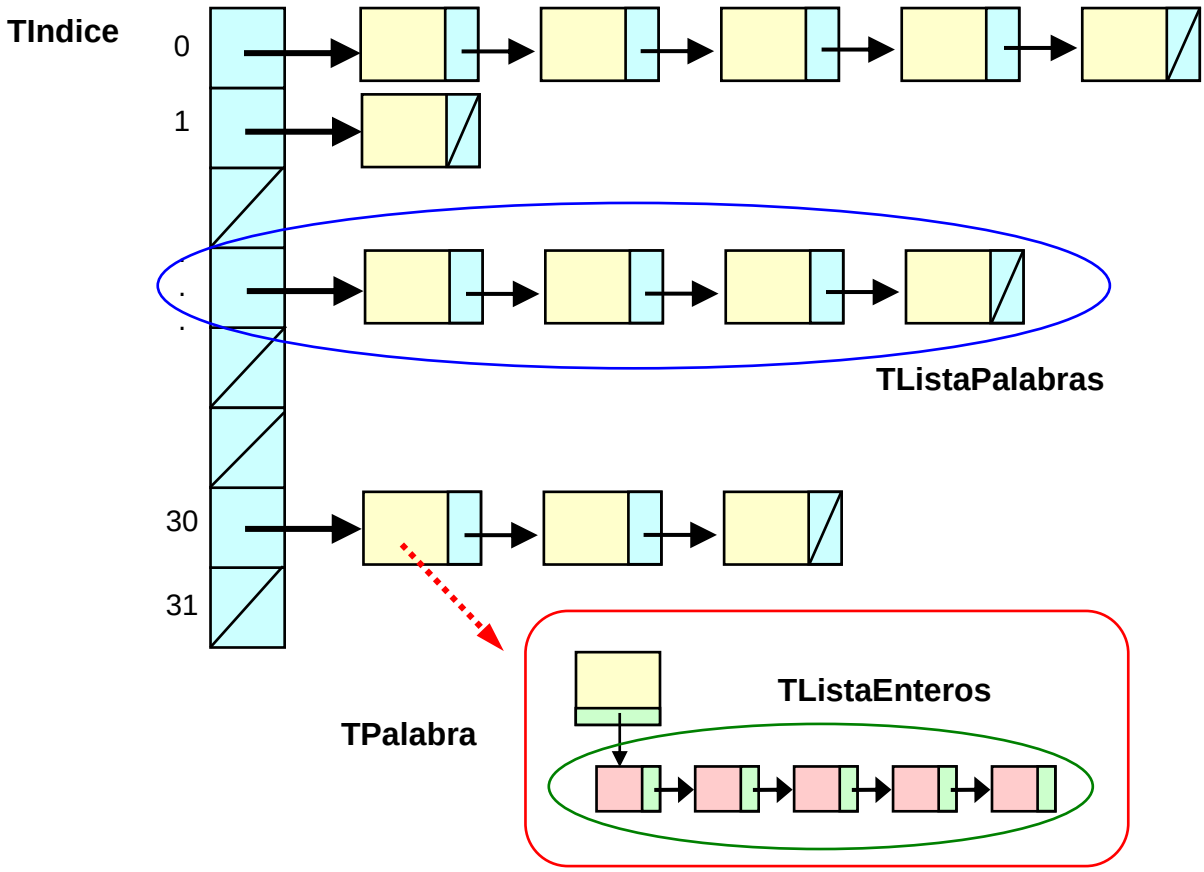
DNI _____ ORDENADOR _____ GRUPO (A/B) _____

Realizar un programa cuyo objetivo será la generación de un índice para un documento de texto.

El índice estará compuesto por una tabla hash de 32 claves con encadenamiento, cuya función hash vendrá determinada por el valor ASCII de la primera letra de la palabra en módulo 32.

El encadenamiento de palabras dentro de una misma clave será implementado utilizando una lista enlazada simple ordenada (por la palabra misma).

Para cada palabra se deberá almacenar tanto su cadena de caracteres como la lista de posiciones que ocupa dentro del fichero (ya que una misma palabra puede aparecer múltiples veces en un mismo documento). La lista de posiciones se mantendrá como una lista enlazada simple ordenada de enteros.



Este programa deberá constar al menos de los siguientes módulos (LA DEFINICIÓN DE LOS MÓDULOS NO SE PODRÁ MODIFICAR):

- **`MError`**
  Tipo enumerado **`TError`** con los posibles valores:

- **Ok** : Operación sin error.
- **EstructuraLlena** : La operación no pudo efectuarse por falta de espacio en la estructura.
- **ElementoInexistente** : La operación no pudo efectuarse por no encontrarse el elemento.
- **ElementoExistente** : La operación no pudo efectuarse porque la estructura ya contiene dicho elemento.

- **MListaEnteros**
  Tipo **TListaEnteros**

```
TListaEnteros CreaListaEnteros ();
/* Función que crea una lista de enteros */
void DestruyeListaEnteros (TListaEnteros & l);
/* Función que destruye una lista de enteros */
void InsertaListaEnteros (TListaEnteros & l, int valor, TError & error);
/* Inserta en orden el nuevo valor. Si el elemento ya se encuentra en la
lista no se modifica la misma y se devuelve el error ElementoExistente */
bool ListaEnterosVacia (TListaEnteros l);
/* Función que indica si la lista está vacía */
bool ListaEnterosLlena (TListaEnteros l);
/* Función que indica si la lista está llena */
void SacaPrimeroListaEnteros (TListaEnteros &l, int &valor, TError &error);
/* Extrae el primer elemento de la lista, devolviendo su valor y eliminándolo
de la misma. Devolverá el error ElementoInexistente si la lista se encuentra
vacía */
```

- **MPalabra**
  Tipo **TPalabra**
  Tipo enumerado **TComparacion** con los posibles valores:
  - **igual** : Las dos palabras son iguales.
  - **mayor** : La primera palabra es mayor en orden alfabético que la segunda.
  - **menor** : La primera palabra es menor en orden alfabético que la segunda.

```
TComparacion ComparaPalabras (TPalabra p1, TPalabra p2);
/* Compara dos palabras indicando si p1 es menor, igual o mayor que p2. La
comparación de palabras será sensible a mayúsculas y minúsculas, y se
realizará atendiendo exclusivamente a la cadena de caracteres que la
consituye, sin tener en cuenta la lista de posiciones en el fichero */
TPalabra CreaPalabra (string palabra);
/* Crea una palabra partiendo de la cadena indicada */
void DestruyePalabra (TPalabra & p);
/* Destruye la palabra indicada */
void NuevaPosicion (TPalabra & p, int posicion, TError & error);
/* Añade la posición indicada en la lista de posiciones */
void Palabra (TPalabra p, string & c);
/* Devuelve la cadena de caracteres de la palabra especificada */
void SacaPrimeraPosicion (TPalabra & p, int & posicion, TError & error);
/* Devuelve la primera posición en la lista, eliminándola de la lista */
```

- **MListaPalabras**
  Tipo **TListaPalabras**

```
TListaPalabras CreaListaPalabras ();
/* Función que crea una lista de palabras */
void DestruyeListaPalabras (TListaPalabras & l);
/* Función que destruye una lista de palabras */
void InsertaListaPalabras (TListaPalabras &l,TPalabra valor,TError &error);
/* Inserta en orden el nuevo valor de palabra. Si ya se encuentra en la lista
no se modifica la misma y se devuelve el error ElementoExistente */
bool ListaPalabrasVacia (TListaPalabras l);
/* Función que indica si la lista está vacía */
```

```
bool ListaPalabrasLlena (TListaPalabras l);
/* Función que indica si la lista está llena */
void SacaPrimeroListaPalabras (TListaPalabras & l, TPalabra & valor,
                               TError & error);
/* Extrae el primer elemento de la lista, devolviendo su valor y eliminándolo
de la misma. Devolverá el error ElementoInexistente si la lista se encuentra
vacía */
```

- **MIndice**
  Tipo **TIndice**

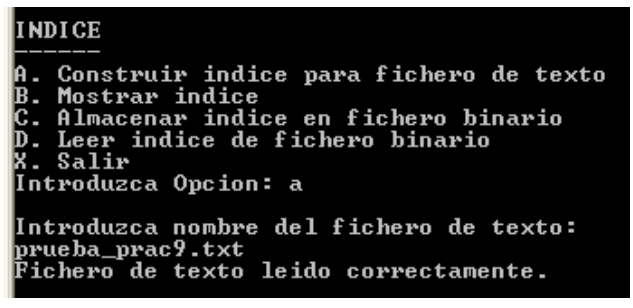
```
void CreaIndice (TIndice & i);
/* Crea un índice vacío */
void DestruyeIndice (TIndice & i);
/* Destruye el índice, dejándolo nuevamente vacío */
void NuevaAparicion (TIndice & i, string palabra, int posicion,
                     TError & error);
/* Añade una nueva aparición en el índice para una palabra determinada. Si la
palabra ya existía incluye la nueva posición en su lista de posiciones.
Únicamente devuelve un error si falta memoria para realizar la operación
(EstructuraLlena) o bien si se intenta insertar una palabra y posición que ya
existía (ElementoExistente) */
void LeeIndiceFichero (ifstream & fi, TIndice & i, TError & error);
/* Destruye el índice actual y lo vuelve a generar a partir del contenido del
fichero binario especificado en el descriptor */
void EscribeIndiceFichero (ofstream & of, TIndice &i);
/* Almacena el índice actual en el fichero binario especificado en el
descriptor. El índice en memoria deberá preservarse tras la llamada */
void EscribeIndicePantalla (TIndice &i);
/* Escribe el índice actual en la pantalla, debiéndose preservar el valor del
índice en memoria tras la llamada */
```

La interfaz del programa se realizará mediante un menú que se detalla seguidamente:

```
INDICE
------
A. Construir indice para fichero de texto
B. Mostrar indice
C. Almacenar indice en fichero binario
D. Leer indice de fichero binario
X. Salir
```

De las opciones del menú anterior, las OPCIONES BÁSICAS que deben realizar TODOS LOS ALUMNOS son:

- **Opción A**: Solicitará el nombre de un fichero de texto, destruirá el índice actualmente en memoria y generará uno nuevo para el archivo de texto especificado.
  En el fichero de texto de entrada se reconocerán aquellas palabras compuestas exclusivamente por letras, cualquier otro carácter se considerará como separador.



- **Opción X**: Destruirá el índice y finalizará la ejecución del programa.

Los ALUMNOS QUE NO TENGAN SUPERADO EL TRABAJO EN CLASE (tienen una nota acumulada < 2), además tendrán que implementar las siguientes opciones del menú principal:

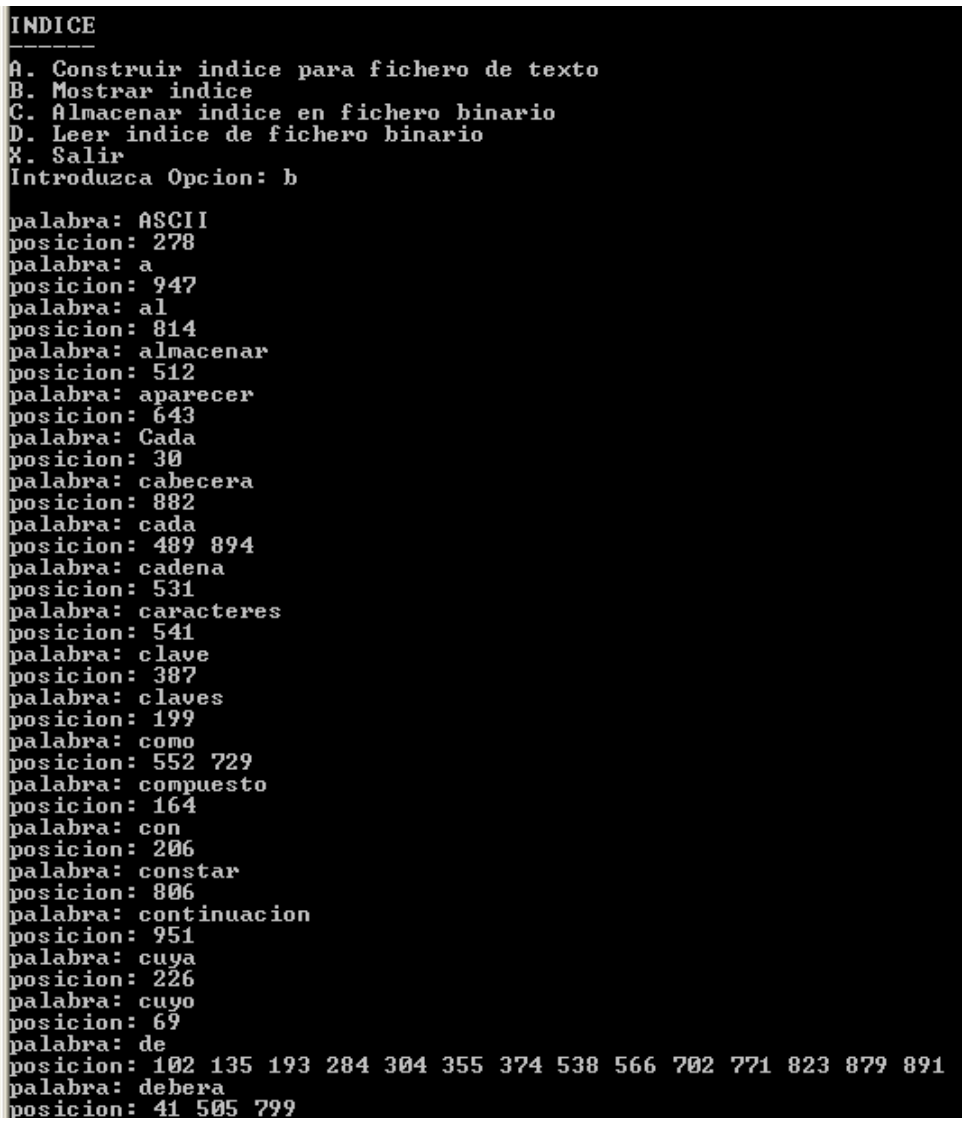
- **Opción B**: Mostrará el índice construido para el archivo.

El formato de salida del índice por pantalla será:

```
palabra: palabra
posicion: posicion1 posicion2 posicion3 ...
```

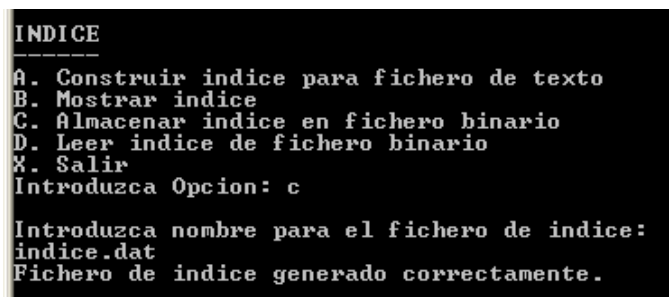
Por ejemplo:

```
palabra: pepe
posicion: 10 45 1245
palabra: juan
posicion: 90
palabra: bueno
posicion: 120 256 423 1045
```



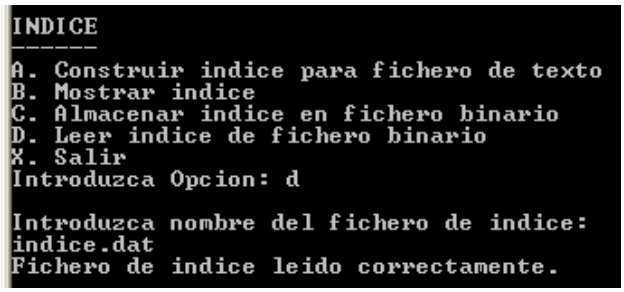
Una vez completadas las operaciones pedidas, PARA SACAR MÁS NOTA se podrán implementar las siguientes opciones adicionales:

- **Opción C**: Almacenará el índice en un fichero binario, cuyo nombre deberá solicitarse al usuario.
  El formato de los ficheros binarios para almacenar los índices es libre, la única restricción es que el programa debe ser capaz de leer los ficheros que almacena.



- **Opción D**: Destruirá el índice actualmente en memoria y generará uno nuevo partiendo de la

información almacenada en el fichero binario cuyo nombre se solicitará al usuario.



NOTAS IMPORTANTES:

- Debido a un fallo del compilador de C++ integrado en Dev-C++, el método `tellg ()`, que permite obtener la posición del cursor en un flujo de entrada, no funciona correctamente en ficheros de texto. Por ello, el fichero de texto de entrada deberá o bien manejarse en formato binario, o bien leyendo el fichero carácter a carácter usando una función del tipo:
  `void LeePalabra (ifstream &fich, string &palabra, int &posIni, int &posFin)`
- Obsérvese también que, a excepción del tipo `TIndice`, no existen funciones para recorrer e imprimir las listas, por lo que para realizar las operaciones de imprimir o escribir en fichero binario listas será necesario extraer e insertar los elementos de las mismas, pudiendo utilizarse las estructuras adicionales para almacenamiento temporal que se crean oportunas, pero de tal forma que al final del proceso las listas queden con el mismo contenido inicial.

CONSIDERACIONES SOBRE EL EXAMEN:

- **Es obligatorio trabajar en una carpeta con nombre LPGES10.**
- El proyecto Dev-C++ deberá denominarse "**jun10_indice.dev**"
- Todo **PROGRAMA QUE NO COMPILE** o tenga **efectos laterales** se considerará **SUSPENSO**.
- Se podrán añadir tipos auxiliares en las cabeceras de los módulos para completar los tipos indicados.
- Queda completamente prohibido añadir ninguna función o procedimiento público en los ficheros de cabecera de los módulos indicados, así como alterar las cabeceras de las funciones especificadas.
- Se podrán añadir cuantas funciones, procedimientos y tipos privados a los módulos que se considere oportunos.
- Se podrán añadir cuantos módulos adicionales se estime oportuno, siempre y cuando no estén destinados a sustituir alguno de los módulos anteriormente indicados.
- El uso de detalles de implementación de un módulo, fuera del módulo de implementación del mismo será **CAUSA de SUSPENSO**, es decir, en el programa principal o en módulos distintos a los mencionados anteriormente no se podrá hacer cosas tipo: l= NULL, ptr=ptr->sig, etc.
- Se recomienda y valora el tratamiento de errores y la buena descomposición del programa principal en procedimientos y funciones, así como el uso de procedimientos y funciones auxiliares dentro de la implementación de los módulos cuando estas sean necesarias.