**Programación en lenguaje c**

**Proyecto N° 1**

Comisión N°24

Feuilles, Juan Cruz

Rayes, Tomas Rodolfo

Diaz, Marianela Lais

**Objetivos del proyecto**

El objetivo principal del proyecto es implementar en lenguaje C un programa que registre la frecuencia de aparición de palabras que forman parte de un archivo de texto. Para ello, se implementa TDA lista, TDA mapeo y Evaluador.

**Detalles de implementación**

***TDA Lista***

Contiene elementos que son punteros genéricos. Es implementada mediante una estructura simplemente enlazada con celda centinela y utilizando el concepto de posición indirecta.

Las funciones que integran al TDA Lista son los siguientes:

*void crear\_lista(tLista \* l):* Inicializa una lista vacía y crea una referencia a ella \*L.

*void l\_insertar(tLista l, tPosicion p, tElemento e):* inserta en la lista, un nuevo elemento “e” en la posición “p”.

*void l\_eliminar(tLista l, tPosicion p, void (\*fEliminar)(tElemento)):* Elimina de la lista el elemento que se encuentra en la posición ”p” y lo hace mediante la función parametrizada “fEliminar”, también libera el espacio que esta ocupa en memoria.

*void l\_destruir(tLista \* l, void (\*fEliminar)(tElemento)):* Destruye la lista eliminando cada uno de los elementos que esta tiene almacenados, mediante la función parametrizada “fEliminar”, también elimina las posiciones en que estos están contenidos y libera el espacio que ocupan cada una en menoria, por ultimo libera la memoria donde se almacena la lista.

*tElemento l\_recuperar(tLista l, tPosicion p):* retorna el elemento que se encuentra en la posición p de la lista.

*tPosicion l\_primera(tLista l):* Retorna la primera posición de la lista “el centinela”.

*tPosicion l\_siguiente(tLista l, tPosicion p):* Retorna la posición siguiente de la posición en la lista pasada por parámetro.

*tPosicion l\_anterior(tLista l, tPosicion p):* Retorna la posición anterior de la posición en la lista pasada por parámetro.

*tPosicion l\_ultima(tLista l):* Retorna la última posición de la lista.

*tPosicion l\_fin(tLista l):* Retorna la posición fin de la lista.

*int l\_longitud(tLista l):* Retorna la longitud de la lista.

***TDA Mapeo***

Utiliza claves y valores que son punteros genéricos. Se implementa mediante una tabla de hash abierta haciendo uso del TDA Lista mencionado anteriormente.

Las funciones que integran el TDA mapeo son los siguientes:

*void crear\_mapeo(tMapeo \* m, int ci, int (\*fHash)(void \*), int (\*fComparacion)(void \*, void \*)):* Inicializa un mapeo con una tabla de capacidad inicial mayor o igual a diez filas. Se referencia al mapeo mediante \*m. Las funciones parametrizadas una calcula el valor hash para una clave y la otra la realiza una comparación entre dos claves.

*void reHash(tMapeo m):* Duplica el tamaño de la tabla e inserta todas las entradas nuevamente acorde a su valor hash actual.

*tValor m\_insertar(tMapeo m, tClave c, tValor v):* Inserta en el mapeo una entrada con clave y valor pasados por parámetros. Si esa clave ya existe dentro del mapeo, se modifica el valor de esta y se retorna el valor anterior, retorna nulo si la entrada no se encontraba dentro del mapeo.

*void m\_eliminar(tMapeo m, tClave c, void (\*fEliminarC)(void \*), void (\*fEliminarV)(void \*)):* Elimina del mapeo la entrada que contiene la clave c, si es que existe. La clave y valor son eliminados mediante las funciones parametrizadas y libera el espacio que esta ocupa en memoria.

*void m\_destruir(tMapeo \*m, void (\*fEliminarC)(void \*), void (\*fEliminarV)(void \*)):* Destruye cada fila “lista” de la tabla, utilizando el destruir de lista y las funciones “fEliminarC” y “fEliminarV”

*tValor m\_recuperar(tMapeo m, tClave c):* Retorna el valor asociado a la clave pasada por parámetro, si es que existe en el mapeo, de lo contrario retorna nulo.

*void fEliminarEntradas* *(tElemento e):* Elimina el valor y la clave de la entrada pasada por parámetro, utilizando “fEliminarC” y “fEliminarV” que fueron dadas tanto por la función eliminar como por el destruir del propio mapeo, esto incluye liberar el espacio que ocupan en memoria la clave, el valor y la entrada.

*void fNoEliminar(tElemento e):* El valor de elemento no es eliminado, esto se usa en reHash para poder usar el destruir de lista en las filas de la tabla original sin perder sus elemento y pasarlos a la nueva tabla.

***Evaluador***

Es la aplicación final con la funcionalidad requerida, se ejecuta bajo línea de comando pasándole el archivo de tipo texto que desea analizar, para esto se vale los TDA Mapeo y Lista.

Recibe el archivo del cual se recuperan las palabras insertándolas a un mapeo, si la palabra no está presente en el mapeo esta se inserta con un valor uno, si la palabra ya está se le incrementa el valor de la entrada correspondiente en uno. Esto permite computar las apariciones de una palabra en el archivo, recurriendo al recuperar de mapeo usando como clave la misma palabra. Al cerrar la aplicación se libera toda la memoria que esta haya reservado.

Como observaciones generales, el comparador utilizado fue el correspondiente a String, y la función hash es una sumatoria de los ordinales con un corrimiento.

**Guía de usuario**

El objetivo de esta guía es dar al usuario una noción (imágenes y escritos) del funcionamiento del programa evaluador y no un manual detallado, el cual procesa las palabras de un archivo de texto registrando la cantidad de apariciones o no de una determinada palabra ingresada por el usuario. Se asume que el usuario sabe utilizar la consola para poder ejecutar el programa.

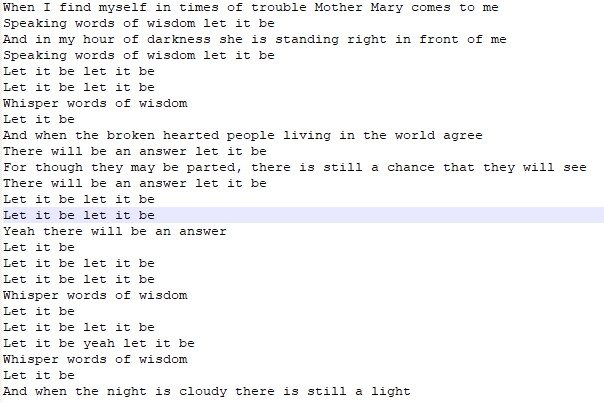
**Concepto de palabra**: El programa considera palabra a carácter o cadena caracteres del código ASCII básico exceptuando espacios, tabulador o fin de línea. Cabe destacar que las letras en su versión mayúscula es distinta que su versión minúscula y viceversa, es decir “A” es distinta de “a”.

**Ejecución del programa**

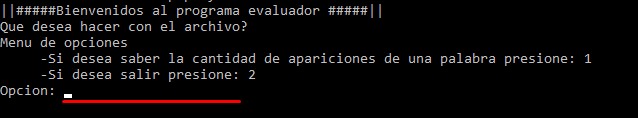
Una vez abierta la consola dirigirse a la ruta donde se encuentra el archivo evaluador.exe y detallando la ruta y nombre del archivo a procesar (ejemplo el archivo “archivo.exe” está en la misma carpeta que el ejecutable para mayor simplificación).

C:\Users\Liche\Desktop\recorrido\guia del usuario\consola abrir .jpg

-A modo de ejemplo el archivo contiene la letra de una canción, dicho contenido se muestra en la siguiente imagen.



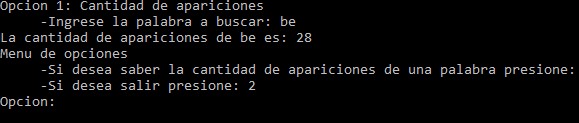
Dentro del programa se procederá a abrir el archivo en modo *solo lectura* y se mostrará un menú de opciones como el que se muestra a continuación esperando a que el usuario ingrese la opción a realizar.



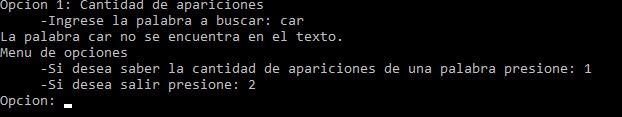
De elegir la opción “1” le pedirá al usuario que palabra desea analizar ejemplo la palabra “be”.

C:\Users\Liche\Desktop\recorrido\guia del usuario\be.jpg

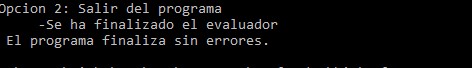
Si la palabra ingresada se encuentra mostrara por pantalla la cantidad de apariciones de la palabra y mostrará de nuevo el menú de opciones. En este caso “be” si se encuentra en el archivo.



De no encontrarse la palabra “be” en el archivo el programa mostrará que la palabra no se encuentra como lo hace con la palabra “car” por ejemplo. Luego mostrará de nuevo el menú de opciones.



De elegir la opción “2”: El programa finalizará liberando previamente la memoria empleada para su funcionamiento y cerrando el archivo utilizado.



En caso de detenerse la ejecución por un erro este será identificado.