Segundo parcial

Parte teórica

1. ¿Cuál es la relación entre punteros y arreglos?
2. ¿Cuál es la relación entre una *unión* y una *struct*?
3. ¿Qué pasa si se invoca la función *free* pasándole un puntero a un arreglo creado estáticamente?
4. Explique cómo se comporta la función *realloc* según los diferentes valores de sus parámetros.
5. ¿Qué son los flujos (*streams*) y para qué sirven?
6. ¿En qué caso/s es necesario manipular programáticamente la estructura FILE?

Parte práctica

En un archivo binario llamado "diccionario.bin" se encuentran almacenadas palabras. El primer campo de tipo entero indica la cantidad total de palabras almacenadas consecutivamente y a continuación, antes de cada palabra y con un campo entero, se indica la cantidad de caracteres (incluye el carácter de final de cadena) que forman la palabra que sigue, y así siguiendo hasta el final del archivo:

Int cantidad total de palabras	
Int cantidad de caracteres(incluye '\0')	Bytes que forman la primer palabra
Int cantidad de caracteres	Bytes que forman la segunda palabra
….	…

Se pide realizar un programa en lenguaje C que permita generar un diccionario inverso. Estos diccionarios se caracterizan por presentar las palabras en orden alfabético ascendente pero observando las palabras desde su último carácter hasta el primero (por ejemplo: hola > aloh).

El programa recibirá dos parámetros como argumentos del main: nombre del archivo binario (fuente) y nombre del archivo de texto de salida (destino). El nombre sugerido para el archivo de salida es "diccionario_inverso.txt". El algoritmo a implementar deberá hacer lo siguiente:

1. Validar cantidad de parámetros del main.
2. Cargar en memoria HEAP la lista de palabras almacenadas en el archivo de entrada. Se sugiere que estos datos se encuentren estructurados en un arreglo de elementos de tipo: typedef struct { int caracteres; char* palabra;} palabras_t;
3. Imprimir en pantalla la lista de palabras del diccionario.
4. Ordenar la lista de palabras para obtener un diccionario inverso.
5. Mostrar en pantalla la lista de palabras ordenadas.
6. Generar el archivo de texto de salida llamado "diccionario_inverso.txt".
7. Liberar la memoria dinámica pedida al sistema operativo.

Aclaraciones

- Lea atentamente el enunciado del ejercicio. Durante la primera h se contestarán las dudas que surjan de los enunciados. Recuerde que primero deberá saber QUE hacer y luego pensar COMO hacerlo.
- Si rinde el examen en una PC de escritorio recuerde guardar el proyecto de CodeBlocks en la unidad D: y que el path que no contenga espacios ni caracteres especiales como la ñ o las vocales acentuadas. Esto último es igual para las notebooks.
- Los prototipos de función y sus implementaciones deberán estar en archivos separados. Por supuesto que el programa deberá compilar sin errores ni advertencias y deberá comportarse de forma predecible durante su ejecución.
- Los nombres del archivo de entrada y salida se deberán suministrar al programa a través de los argumentos de la función principal *main*().
- Se debe liberar toda la memoria pedida en el HEAP (a cada MALLOC le corresponde un FREE).
- Se debe usar la función ***qsort*** para ordenar la lista de palabras del diccionario.
- No se puede usar la función de librería ***strrev***() de la librería estándar, esta funcionalidad debe ser implementada por el alumno.
- Se permite usar la función de libería ***strncmp***().
- La función de impresión en pantalla y en el archivo de texto debe ser reutilizada, es decir, se debe usar la misma función para ambas salidas.
- La salida del programa deberá ser como muestra la imagen que sigue:

```
Diccionario normal
6-adios
7-bombon
8-geranio
5-hola
5-rosa
8-tractor

Diccionario inverso
5-hola
5-rosa
7-bombon
8-geranio
8-tractor
6-adios
```

Exitos! =)