

Práctica 5

Archivos de texto y archivos binarios

- 1) El siguiente código lee la información de un archivo de texto y la imprime en la pantalla. Analice, compile y ejecute el programa. Utilice el archivo “prueba.txt” provisto por la cátedra.

```
#include <stdio.h>
#include <stdlib.h>
#define LONG 300

int main() {
    FILE *f;
    char linea[LONG];

    // Abrir el archivo
    f = fopen("prueba.txt", "r");

    if (f == NULL) {
        printf ("\nError al abrir archivo fuente\n");
        return 1;
    }

    while (!feof(f)) {
        // leer una línea del archivo
        fgets (linea, LONG, f);
        // imprimir la línea en la pantalla
        puts (linea);
    }

    fclose(f);
    return 0;
}
```

¿El código funciona correctamente? Si no es así, corrijalo para que lo haga.

Nota: Preste atención al formato del archivo. La función *fgets()* procesa hasta encontrar un salto de línea (el cual es retenido). Si luego del último renglón no hubiese un salto de línea, el archivo no tendría un formato válido para procesar.

- 2) Escriba un programa que copie el contenido de un archivo de texto en otro nuevo.
- a) Utilizando las funciones *fgetc* y *fputc*.
 - b) Utilizando las funciones *fgets* y *fputs*.
 - c) Utilizando las funciones *fread* y *fwrite*.
- 3) Escriba un programa que procese un archivo de texto e informe la cantidad de caracteres minúsculas, mayúsculas y dígitos que posee.
- 4) Se desea leer y procesar información de precipitaciones del mes de enero. Para ello se dispone de un archivo de texto (llamado *precipitaciones.txt*) con el siguiente formato:

0-2-0-0-7-22-11-0- ... -0-

Por cada de los 31 días se tiene un número entero indicando los milímetros llovidos, seguido de un guión medio (-) como delimitador. Escriba un programa que lea la información del archivo y derermine el día con mayor precipitación. Para evaluar el programa, genere un archivo con el formato establecido utilizando un editor de texto plano (por ejemplo: *Bloc de notas* o *Notepad++*).

Nota: puede utilizar la función ***fscanf*** para procesar cada valor de precipitación.

- 5) Se desea leer y procesar información de un listado de apuestas. Para ello se dispone de un archivo de texto (llamado `apuestas.txt`) con el siguiente formato:

`código_de_apuesta | monto_apostado; ... ; código_de_apuesta | monto_apostado;`

Donde cada apuesta se compone de un número entero (código de apuesta) y un número flotante (monto apostado). Escriba un programa que procese la información del archivo e informe el monto total apostado.

Ejemplo: `1 | 100.0; 65 | 50.5; 23 | 34.5;` debe informar: “El monto total apostado es \$185”.

Nota: Intente resolver el problema leyendo cada tupla (código, monto) al mismo tiempo.

- 6) Escriba un programa que permita a un usuario consultar si un conjunto de palabras existen o no en un diccionario. El usuario ingresa de a una palabra y la consulta finaliza cuando ingresa la palabra “ZZZ”. Para cada palabra ingresada se debe informar si la misma pertenece o no al diccionario.

El diccionario consiste en un archivo de texto y las palabras se encuentran ordenadas en forma ascendente (una por línea). Se desea generar una estructura de datos dinámica (memoria RAM) en la cual se almacenen las palabras de todo el diccionario. Luego, verifique la pertenencia de las palabras ingresadas por el usuario utilizando dicha estructura en lugar del archivo.

- 7) Escriba un programa que lea desde teclado números enteros de una cifra (del 0 al 9) y que cree dos archivos, los cuales almacenen los números leídos. El primer archivo debe llamarse “`numeros.txt`” y contener en formato texto todos los números leídos en forma consecutiva. El segundo archivo debe llamarse “`numeros.dat`” y contener todos los números leídos en formato binario.

- Una vez ejecutado el programa, utilice un editor de texto (como por ejemplo: el *Bloc de notas* en *MS Windows*, o *Gedit* en *GNU/Linux*) para abrir el archivo de texto y corroborar que la información almacenada sea la correcta. Usando el mismo editor, intente abrir el archivo binario. ¿Es posible visualizar los números? ¿Por qué cree que no es posible?
- Compare los tamaños que ocupan cada archivo. ¿Cuál es más grande? ¿En qué ocasiones cree usted que sería mejor utilizar cada tipo de archivos?

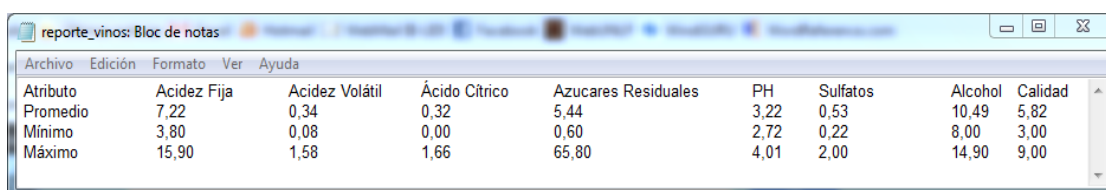
- 8) Escriba un programa que lea información de 20 jugadores de tenis. De cada jugador se lee nombre y apellido, edad, cantidad de títulos, ranking actual y fortuna acumulada. Defina una estructura de datos adecuada para la información y almacene la misma en un archivo binario. Finalizada la lectura, procese los datos almacenados en el archivo e informe:

- a) Nombre y apellido del jugador con mejor ranking.
- b) Nombre y apellido del jugador que más títulos ha ganado.

- 9) Escriba un programa que abra el archivo binario generado en el ejercicio anterior y que imprima en pantalla el tamaño en bytes de dicho archivo. ¿Por qué cree Ud. que tiene ese tamaño?

- 10) Un archivo *csv* (Comma Separated Values) contiene información separada por comas. Cada renglón (fila) contiene un registro de información. Cada columna contiene un campo particular de información. La primera fila es especial ya que contiene los nombres de los campos.

Se desea obtener información a partir del archivo llamado *vinos.csv* (el cual se encuentra en la Sección “Ing. Gral. y Contenidos” del curso de la cátedra en WebUNLP). El programa debe generar un archivo de texto con un resumen que indique el valor máximo, mínimo y promedio para cada uno de los campos del archivo. Este archivo debe llamarse *reporte_vinos.txt* y debe guardarse en la misma ruta que el archivo *vinos.csv*. El formato del archivo debe ser similar al del siguiente ejemplo:



Atributo	Acidez Fija	Acidez Volátil	Ácido Cítrico	Azúcares Residuales	PH	Sulfatos	Alcohol	Calidad
Promedio	7,22	0,34	0,32	5,44	3,22	0,53	10,49	5,82
Mínimo	3,80	0,08	0,00	0,60	2,72	0,22	8,00	3,00
Máximo	15,90	1,58	1,66	65,80	4,01	2,00	14,90	9,00

- 11) a. Realice un programa que permita generar un índice para acceder por DNI de manera eficiente a la información del archivo de texto “personas.csv”. Este archivo contiene los datos de una persona (identificador, dni, nombre, apellido, correo, ciudad, país y trabajo que desarrolla) en formato CSV. El índice debe generarse en memoria y estar ordenado por DNI para finalmente almacenarlo en el archivo binario “personas.idx”. Este archivo binario deberá tener por cada línea del archivo “personas.csv” una entrada que contenga el DNI de la persona junto con la posición absoluta dentro del archivo a los datos asociados a dicho DNI. Diseñe una estructura de datos adecuada para el índice y tenga en cuenta al momento de generarlo que la primera línea del archivo CSV es de encabezado.

A continuación puede verse un ejemplo para 5 personas:

byte inicial p/c fila en CSV	Ejemplo de 5 primeras filas del archivo “personas.csv”							
0	id	dni	apellido	nombre	trabajo	correo	ciudad	país
51	1	30919537	Alldridge	Enoch	Auditor	Enoch_Alldridge440@deons.tech	Worcester	Vietnam
135	2	50696081	Doherty	Noah	Treasurer	Noah_Doherty3511@deons.tech	Venice	Marshall Islands
222	3	41858450	Gates	Rae	HR Coordinato	Rae_Gates7739@svelto.biz	Memphis	Cameroon
301	4	57597223	Forth	Nicholas	Lecturer	Nicholas_Forth1084@supunk.biz	Boston	Uzbekistan
385	5	12325238	Ingram	Faith	Health Educato	Faith_Ingram6155@vetan.org	Seattle	Bolivia

Ordenando por DNI las 5 primeras entradas, el orden sería 5,1,3,2 y 4 (columna id). Por lo tanto el archivo binario “personas.idx” debería quedar:

DNI	Desplazamiento dentro de “personas.csv”
12325238	385
30919537	51
41858450	222
50696081	135
57597223	301

- b. Realice un programa que muestre en pantalla toda la información de una persona a través de un DNI ingresado por teclado. Si la persona no existe deberá informar el error.

Tenga en cuenta para la implementación:

- Utilizar una estructura de datos adecuada que permita tener el índice completamente en memoria.
- Implemente para la búsqueda una función que realice una búsqueda dicotómica.
- Implemente el siguiente prototipo para la función de búsqueda:
`int buscar(FILE* personas, Indice indice, int dni, persona * persona)`
donde:
 - **personas** es el descriptor del archivo “personas.csv”
 - **indice** es la estructura de datos con la información del archivo “personas.idx”.
 - **dni** es el nro de dni de la persona a buscar
 - **persona** es una estructura con los datos con la información asociada al dni de la persona.
 - El valor de **retorno** es 0 sino encuentra a la persona y 1 en caso contrario.

- c. Reimplemente b) para utilizar en la función de búsqueda el descriptor del archivo índice. Además modifique el valor de retorno para reflejar la cantidad de veces que se accedió al disco durante la búsqueda en caso de que lo encuentre. El prototipo de la función de búsqueda quedaría:

`int buscar(FILE* personas, FILE* indice, int dni, persona * persona)`