PRÁCTICAS DEL LENGUAJE C

BÁSICOS:

1. Un programa que imprima la tabla de códigos ASCII., mostrará la tabla en número de columnas y podemos hacer una pausa por un determinado número de filas. Con la función getch().

2. Programa que imprima por pantalla todos los números primos del 1 al 100.

ARRAYS:

3. Comprobar si un array de número está ordenado ascendentemente. Declarar los vectores con sus elementos no leerlos de pantalla.

4. Comprobar si los elementos de un array forman un número capicúa. Declarar los vectores con sus elementos no leerlos de pantalla.

5. Implementar la función getsn(char *, int), esta función lee caracteres hasta que se llene o hasta que pulsemos el enter, el char 13, para leer carácter a carácter vamos a usar getche() lee y hace eco en pantalla. El main tendrá el siguiente esquema, define un array de char de tamaño MAX y llama a la función getsn después muestra por pantalla la cadena leída y pregunta al usuario si quiere seguir introduciendo cadenas.

RECURSIVIDAD:

6. Convertir un número a binario, imprimir el resultado por pantalla. Para probarlo imprimir por pantalla los 256 primeros números en binario: del 0 al 255.

7. Modificar la función anterior para que nos valga con otras bases: Para que la llamada a la función sea de la forma: *cambioDeBase(numero, base);*

8. printer(const char *); Imprime una cadena de caracteres.

9. strlen(const char *), Utilizar el operador ternario **? :**

10. strcmp(const char *, const char *)

11. reverse(const char *) → Imprime una cadena al revés.

12. putsn(const char *, int) → Imprime de forma recursiva los n primeros caracteres de la cadena que reciba por argumento.

MATRICES:

13. Definir una matriz de 3 x 3 y con sus valores constantes, no pedir los datos por pantalla, calcular la suma de los elementos de la diagonal principal.

14. Definir una matriz de enteros, tiene que ser cuadrada, a partir de una constante. Se rellenará con números aleatorios 0, 1, 2 y tenemos que comprobar si trazamos una línea por la diagonal principal de la matriz que los números sean iguales. También que imprima la tabla por pantalla.

1	**0**	**2**
0	**1**	**1**
2	**1**	**1**

FICHEROS:

15. Implementar un programa C que realice una copia de un fichero, recibirá por los argumentos de main el nombre del fichero origen y el nombre del fichero destino.

16. Definir una estructura para almacenar la información de una persona: nombre, apellidos, dni y la fecha de nacimiento que será otra estructura

representada por día, mes y año. Pedir los datos por pantalla y el número de personas que queremos almacenar y cuando termine el usuario se grabarán los datos en un fichero binario.

17. Una vez creado el fichero, hacer consultas en el fichero, mostrando el registro que indique el usuario, mediante un número que irá de 0 a N-1.

LIBRERIAS ESTÁTICAS:

18. Implementar una librería estática, con las funciones recursivas de tratamiento de cadenas recursivas. Definir los prototipos en un .h, generar la librería y luego en un programa C probar las distintas funciones.

MEMORIA DINÁMICA:

19. El usuario teclea un número indeterminado de frases por teclado, tenemos que almacenarlas en memoria dinámica, tanto en número de frases como la longitud de las mismas. Cuando el usuario teclea un ***** el programa termina:
    a. Muestra un listado de todas las frases tecleadas.
    b. Las graba en un fichero de texto, llamado salida.txt. OJO, una frase por línea.
    c. Después hay que liberar toda la memoria ocupada.

ESTRUCTURAS DINÁMICAS:

20. Implementar un menú que muestre lo siguiente:

    MENU PRINCIPAL

    1.- Insertar datos.

    2.- Listar cuentas.

    3.- Buscar cuenta.

    4.- Borrar cuenta.

    5.- Borrar todas las cuentas.

    6.- Salir

- Se mantendrá la información en una **lista lineal.**
- La parte de los datos contendrá los siguientes campos:
char *nroCuenta, char *DNI, float saldo.

- Los prototipos de todas las funciones, irán en el fichero: prototipos.h
- La definición de estructuras ira en el fichero: estructuras.h
- Main.c se implementarán las funciones:
    - main() → se encarga de mostrar el menú y gestionar las opciones.
    - menu() → Muestra el menú por pantalla.
    - leerCuenta() → Lee los datos de la cuenta de teclado.
    - imprimeCuenta() → Muestra los datos de la cuenta por pantalla.
    - listarCuentas() → Lista todas las cuentas.
    - insertarDatos() → Llama a leer cuenta y luego añade la cuenta a la lista.
    - añadeCuenta() → Función recursiva que añade los datos de la cuenta en la lista, al final.
    - buscarCuenta() → Pide un número de cuenta por pantalla y llama a otra función recursiva que devolverá NULL si no encuentra la cuenta y si no mostrará los datos de la misma.
    - borrarCuenta() → Idem de la anterior pero borrando, pero no es recursiva.
    - borrarCuentas() → Libera la memoria de todas las cuentas.

21. Implementar un menú para gestionar un **árbol binario de búsqueda**:
    MENÚ PRINCIPAL
    1.- Añadir un número.
    2.- Los 3 recorridos.
    3.- Buscar un elemento.
    4.- Liberar todos los nodos.
    5.- Calcular la altura.
    6.- Salir.

Los elementos del árbol pueden ser **int**. Seguir los mismos criterios de estructurar los ficheros que en la lista lineal.

22. Implementar una **cola**, con las funciones de encolar (añadir un nuevo elemento) y la función desencolar (sacar un elemento). Será de números mantendremos dos punteros uno a la cabecera y otro al final. Los números se cargarán a partir de un fichero de texto, en el que cada número vendrá en una línea. Se encolarán todos los números y luego se desencolaran imprimiéndolos por pantalla.

DLLs:

23. Implementar una DLL y una aplicación a parte para probarla. Hacer las mismas funciones que con la librería estática.

ARRAYS DE BITS

24. Código HAMMING mediante un array de bits. Montar la trama a transmitir, es decir, con los bits de paridad.

http://es.wikipedia.org/wiki/C%C3%B3digo_Hamming

```
#define MAXBITS 80


typedef unsigned char BYTE;

BYTE arraybits[MAXBITS/8+2];
BYTE Nrobits;


void putbit(BYTE, int);
int getbit(BYTE);
void atob(char *);
void imprimebits(void);
BYTE calculaparidad(void);
void inicializa(void);
BYTE pot2(BYTE);
void insertaBit(BYTE, int);
void Hamming(void);
BYTE NroBitsParidad(BYTE);
void marcaBits(void);
```