

X

X

 $\wedge$ 

```

* Nombre:
*   Stefan Trifan
*
* Hora Inicio:
*   09: 30
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define DIRECCION "precipitacionesMadrid2024.txt"
#define TAM_BLOQUE 10
#define CANTIDAD_MESES 12

#define RED "\033[1;31m"
#define GREEN "\033[1;32m"
#define YELLOW "\033[1;33m"
#define RESET "\033[0m"

// Funciones del programa
char* leeLineaDinamicaFichero(FILE *fd;
float obtenerPrecipitacion(char *cadena);

```

```

// Inicializamos una nueva instancia de la estructura. Reservamos memoria
// Funciones auxiliares
void clearBuffer();

/*
_____|
Inicio main*/

int main(int argc, char *argv[])
{
    printf("_____|_START\n\n");

    // Declaración de variables
    FILE *fd = fopen(DIRECCION4, "r+");
    float valores_precipitaciones[CANTIDAD_MESES] = {0};
    char *cadena;
    float precipitacion_min = 0,
          precipitacion_med = 0,
          precipitacion_max = 0;

    // Comprobamos que el archivo se ha abierto bien
    if(fd == NULL)
    {
        printf("\033[1;31mERROR: No ha sido posible encontrar el fichero\n\033[0m");
        printf("\033[31m\n_____|_FAIL\n\n\033[0m");
        return EXIT_FAILURE;
    }
    else
    {
        printf("\033[1;32mArchivo abierto correctamente\n\n\033[0m");
    }

    // Extraer del fichero los valores mensuales de precipitaciones.
    for(int i = 0; i < CANTIDAD_MESES + 1; i++)
    {
        // Vamos guardando uno por uno los valores de las precipitaciones
        // en nuestro array de flotantes
        if(i >= 1)
        {
            valores_precipitaciones[i - 1] = obtenerPrecipitacion(leeLineaDinamicaFichero(fd));
        }
        else if(i == 0)
        {
            cadena = leeLineaDinamicaFichero(fd); // Condicion para que ignore la primera linea
        }
    }

    // Ordenamos los valores del array de menor a mayor mediante el algoritmo de ordenacion BURBUJA
    for(int i = 0; i < CANTIDAD_MESES - 1; i++)
    {
        for(int j = 0; j < CANTIDAD_MESES - 1 - i; j++)
        {
            if(valores_precipitaciones[j] > valores_precipitaciones[j + 1])
            {
                float temp = valores_precipitaciones[j];
                valores_precipitaciones[j] = valores_precipitaciones[j + 1];
                valores_precipitaciones[j + 1] = temp;
            }
        }
    }

    // Calcular el mes de precipitaciones máximas, mínimas y la media de precipitaciones anuales.
    precipitacion_min = valores_precipitaciones[0];
    precipitacion_max = valores_precipitaciones[CANTIDAD_MESES - 1];
    for(int i = 0; i < CANTIDAD_MESES; i++)
    {
        precipitacion_med += valores_precipitaciones[i];
    }
    precipitacion_med /= CANTIDAD_MESES;

    // Presentar por pantalla cada uno de los valores calculados anteriormente
    printf
    {
        "Mes de máxima precipitación: \033[1;32m %.2f l/m\n\033[0m\n"
        "Mes de mínima precipitación: \033[1;32m %.2f l/m\n\033[0m\n"
        "Media anual de precipitaciones:\033[1;32m %.2f l/m\n\033[0m\n"
        ,precipitacion_max, precipitacion_min, precipitacion_med
    };

    // Agregar al final del fichero las líneas con los datos estadísticos calculados
    // (Mes con precipitaciones máximas, mínimas y media de precipitaciones).
    fprintf
    {
        fd,
        "\n"
        "Mes de máxima precipitación: %.2f l/m\n"
        "Mes de mínima precipitación: %.2f l/m\n"
        "Media anual de precipitaciones: %.2f l/m\n"
        ,precipitacion_max, precipitacion_min, precipitacion_med
    };

    // Cerrar fd
    fclose(fd);

    printf("\n_____|_END\n\n");
    return EXIT_SUCCESS;
}

/*
_____|
Inicio declaracion de funciones */

// Funciones del programa
/**
 * Función que de una línea que se le pasa como
 * parámetro se extrae el valor las precipitaciones
 */
float obtenerPrecipitacion(char *cadena)
{
    // Declaracion de variables
    float valor_precipitacion;

    // Cuerpo funcion
    strtok(cadena, ";");
    valor_precipitacion = atof(strtok(NULL, " ("));

    return valor_precipitacion;
}

// Funciones auxiliares


```


```
/**
 * Devuelve una línea del fichero que se pasa como
 * parámetro utilizando la función fgets o fgetc y
 * reservando memoria dinámica
 */
char *leeLineaDinamicaFichero(FILE *fd)
{
    // Declaración de variables
    char *destino = (char *)malloc(sizeof(char) * TAM_BLOQUE);
    int memoria_actual = TAM_BLOQUE;
    int i = 0;
    char c;

    // Cuerpo función
    while(((c = fgetc(fd)) != EOF) && (c != '\n'))
    {
        if(i == memoria_actual - 1)
        {
            memoria_actual += TAM_BLOQUE;
            destino = (char *)realloc(destino, sizeof(char) * memoria_actual);
        }
        destino[i] = c;
        i++;
    }
    destino[i] = '\0';

    return destino;
}

void clearBuffer()
{
    while(getchar() != '\n');
}
```

 precipitacionesMadrid2024.txt ...

 main.c ...

## 2 ENSAYO

### Censo Animales

Se desea realizar el censo de especies animales de una determinada zona.

Se desea realizar este censo a partir de un programa de manera que el usuario introduzca la información de cada especie por el teclado y sea almacenada en un fichero cuyo nombre es introducido por parámetros del main.

Los datos de cada especie que se almacenarán en la estructura datosEspecie\_t son:

- Nombre Común. Nombre común (ej. Lobo ibérico)
- Grupo taxonómico: Clasificación general. Solo se tendrán en cuenta los siguientes MAMIFERO, AVE, REPTIL, DESCONOCIDO.
- Cantidad de ejemplares: Número de individuos censados.

El programa tendrá como requisitos:

- El nombre del fichero con los datos de las especies censadas se le pasarán al programa por parámetros del main.
- Los datos de cada especie son introducidos por teclado por el usuario sin conocer el número de especies total que se van a introducir. Cuando se pida los datos de "Grupo taxonómico", el usuario debe introducir "MAMIFERO, AVE o REPTIL". En caso de no hacerlo (cualquier otra combinación de caracteres), el valor se considerará "DESCONOCIDO" y el programa debe volver a pedirlo hasta que introduzca un grupo válido. Para comprobar grupos taxonómicos válidos, se recomienda usar una variable global como la siguiente:  

```
char* grupoStr[] = { "MAMIFERO", "AVE", "REPTIL", "DESCONOCIDO" };
```
- Igualmente, cuando se pida la cantidad, el usuario debe introducir un valor numérico válido. En caso contrario, se debe volver a pedir hasta que introduzca un número válido.
- El uso de memoria deberá optimizarse.
- El programa debe pedir datos de cada especie hasta que el usuario lo desee (preguntar si debe continuar después de haber leído una especie correctamente).
- Una vez que el usuario haya determinado no seguir introduciendo datos (el usuario selecciona que no quiere introducir más), el programa debe presentar por pantalla todas las especies introducidas en esa sesión en orden inverso (primero aparecerá la última especie introducida).
- Una vez presentadas, se añadirán al fichero pasado por parámetros del main con el siguiente formato, y se cerrará para salvar su contenido:  

```
<Nombre común>;<grupo taxonómico>;<cantidad de ejemplares>
```

Se pide implementar las siguientes funciones:

```
// Devuelve la cadena de caracteres que el usuario ha //escrito en el teclado por terminal
// reservando memoria de manera dinámica
char* leeLineaDinamica();

// Función que, dada una cadena, compara con todos
// los grupos taxonómicos válidos, y devuelve el
// enumerado correspondiente.
// Si no es un grupo válido, devuelve el valor
// enumerado "DESCONOCIDO"
grupo_e stringAGrupo(char* grupo)

// Función que pide datos de una especie por terminal
// Rellena un elemento de tipo "especie_t" con
// datos leídos y lo devuelve por parámetros de return
especie_t pideEspecie()
```

Un ejemplo de ejecución puede ser:

```
Introduzca nombre de especie:
Leon
Introduzca grupo:
MAMIFERO
Grupo no válido
Introduzca grupo:
MAMIFERO
Introduzca número de individuos:
12
Cantidad no válida
Introduzca número de individuos:
12
Desea salir? Si/No
No
Introduzca nombre de especie:
Aguila
Introduzca grupo:
AVE
Introduzca número de individuos:
```

```
34
Desea salir? Si/No
No
Introduzca nombre de especie:
Iguana
Introduzca grupo:
Reptil
Grupo no válido
Introduzca grupo:
REPTIL
Introduzca número de individuos:
23
Desea salir? Si/No
Si
Nombre: Iguana
Grupo: REPTIL
Numero: 23
Nombre: Aguila
Grupo: AVE
Numero: 34
Nombre: Leon
Grupo: MAMIFERO
Numero: 12
```

## Respuesta

```
/**
 * Nombre:
 *   Stefan Trifan
 *
 * Hora Inicio:
 *   09:30
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TAM_BLOQUE 10

#define RED "\033[1;31m"
#define GREEN "\033[1;32m"
#define YELLOW "\033[1;33m"
#define RESET "\033[0m"

char* grupoStr[] = { "MAMIFERO", "AVE", "REPTIL", "DESCONOCIDO" };

typedef enum
{
    MAMIFERO, AVE, REPTIL, DESCONOCIDO
}
grupo_e;

typedef struct
{
    char *nombre_comun;
    grupo_e grupo_taxonomico;
    int cantidad_ejemplares;
}
contenedor_especie_t; // "contenedor_especie_t" == "especie_t"

// Funciones del programa
grupo_e stringAGrupo(char *grupo);
contenedor_especie_t pideEspecie();

// Funciones auxiliares
int pedirEntero();
char *leeLineaDinamica();
void clearBuffer();

/* =====
Inicio main*/

int main(int argc, char *argv[])
{
    printf("=====START\n\n");

    // El nombre del fichero con los datos de las especies censadas
    // se le pasará al programa por parámetros del main.
    if(argc != 2)
    {
        printf("\033[1;31mUSO: ./main <nombre_archivo.txt\033[0m");
        printf("\033[31m\n=====FAIL\n\n\033[0m");
        return EXIT_FAILURE;
    }

    // Declaracion de variables
    FILE *fd = fopen(argv[1], "w+");
    contenedor_especie_t *c_especies = (contenedor_especie_t *)malloc(sizeof(contenedor_especie_t));
    char *salir;
    int num_especies = 0;

    // Comprobamos que el archivo se ha creado con éxito
    if(fd == NULL)
    {
        printf("\033[1;31mERROR: No ha sido posible crear el fichero\n\033[0m");
        printf("\033[31m\n=====FAIL\n\n\033[0m");
        return EXIT_FAILURE;
    }
    else
    {
        printf("\033[1;32mArchivo abierto correctamente\n\n\033[0m");
    }

    // Pedir datos
    do
    {
        // Llamada a la función pideEspecie()
        c_especies[num_especies] = pideEspecie();

        // El programa debe pedir datos de cada especie hasta que el usuario lo desee
        // (preguntar si debe continuar después de haber leído una especie correctamente
        printf("Quieres seguir introduciendo especies?\n-> ");
        do
        {
            salir = leeLineaDinamica();
        }
    }
    while (salir != "n");
}
```

```

    salir = leeLineaDinamica();
    if((strcmp(salir, "si") != 0) && (strcmp(salir, "no") != 0))
    {
        printf(YELLOW"Introduzca si/no\n"RESET);
    }
}
while ((strcmp(salir, "si") != 0) && (strcmp(salir, "no") != 0));

// Si se quiere seguir ampliando la memoria
if((strcmp(salir, "si") == 0))
{
    // Incrementamos el numero de animales
    num_especies++;
    c_especies = (contenedor_especie_t *)realloc(c_especies, sizeof(contenedor_especie_t) * (num_especies + 1));
}
}
while (strcmp(salir, "si") == 0);

// Se añadirán al fichero pasado por parámetros del main con el siguiente formato
// <Nombre común><grupo taxonómico><cantidad de ejemplares>
for(int i = 0; i <= num_especies; i++)
{
    if(c_especies[i].grupo_taxonomico == 0)
    {
        fprintf
        (
            fd,
            "%s:MAMIFERO:%d\n",
            c_especies[i].nombre_comun,
            c_especies[i].cantidad_ejemplares
        );
    }
    else if (c_especies[i].grupo_taxonomico == 1)
    {
        fprintf
        (
            fd,
            "%s:AVE:%d\n",
            c_especies[i].nombre_comun,
            c_especies[i].cantidad_ejemplares
        );
    }
    else if (c_especies[i].grupo_taxonomico == 2)
    {
        fprintf
        (
            fd,
            "%s:REPTIL:%d\n",
            c_especies[i].nombre_comun,
            c_especies[i].cantidad_ejemplares
        );
    }
}

// Una vez que el usuario hayamos determinado no seguir introduciendo datos
// el programa debe presentar por pantalla todas las especies introducidas
// en esa sesión en orden inverso.
printf("\n\n");
int contador_inverso = num_especies;
for(int i = 0; i <= num_especies; i++)
{
    printf("Nombre: %s\n", c_especies[contador_inverso].nombre_comun);

    if(c_especies[contador_inverso].grupo_taxonomico == 0)
    {
        printf("Grupo: MAMIFERO\n");
    }
    else if (c_especies[contador_inverso].grupo_taxonomico == 1)
    {
        printf("Grupo: AVE\n");
    }
    else if (c_especies[contador_inverso].grupo_taxonomico == 2)
    {
        printf("Grupo: REPTIL\n");
    }

    printf("Numero: %d\n\n", c_especies[contador_inverso].cantidad_ejemplares);

    contador_inverso--;
}

// Liberar memoria y cerrar fd para salvar su contenido:
fclose(fd);
for(int i = 0; i < num_especies + 1; i++)
{
    free(c_especies[i].nombre_comun);
}
free(c_especies);

printf("\n_____END\n\n");
return EXIT_SUCCESS;
}

/* _____
Inicio declaracion de funciones */

// Funciones del programa
/**
 * Función que pide datos de una especie por terminal
 * Rellena un elemento de tipo "contenedor_especie_t" con
 * datos leídos y lo devuelve por parámetros de return
 */
contenedor_especie_t pideEspecie()
{
    // Declaracion de variables
    contenedor_especie_t nueva_especie;
    char *temp;

    // Rellenamos los miembros con los datos que menciona el usuario
    printf("Introduzca nombre de especie:\n-> ");
    nueva_especie.nombre_comun = leeLineaDinamica();

    // En grupo taxonomico el usuario debe introducir la palabra correspondiente
    // Comparamos la cadena con dowhile con la variable global
    // Llamamos a la funcion stringAGrupo(char *grupo)
    do
    {
        printf("Introduzca grupo grupo taxonomico:\n-> ");
        temp = leeLineaDinamica();
        nueva_especie.grupo_taxonomico = stringAGrupo(temp);
        if(nueva_especie.grupo_taxonomico == DESCONOCIDO)
        {
            printf(YELLOW"ALERTA! Grupo taxonomico no existe\n"RESET);

```

```

        printf(YELLOW"Escribe 'MAMIFERO, AVE: O REPTIL': con mayusculas\n"RESET);
    }
}

while (nueva_especie.grupo_taxonomico == DESCONOCIDO);

// Igualmente, cuando se pida la cantidad, el usuario debe introducir un valor numérico válido.
// En caso contrario, se debe volver a pedir hasta que introduzca un número válido.
printf("Introduzca la cantidad\n-> ");
nueva_especie.cantidad_ejemplares = pedirEntero();

return nueva_especie;
};

/**
 * Función que, dada una cadena, compara con todos
 * los grupos taxonómicos válidos, y devuelve el
 * enumerado correspondiente.
 * Si no es un grupo válido, devuelve el valor
 * enumerado "DESCONOCIDO"
 */
grupo_e stringAGrupo(char *grupo)
{
    // Declaración de variables
    grupo_e nuevo_grupo;

    if(strcmp(grupo, grupoStr[0]) == 0)
        nuevo_grupo = MAMIFERO;
    else if(strcmp(grupo, grupoStr[1]) == 0)
        nuevo_grupo = AVE;
    else if(strcmp(grupo, grupoStr[2]) == 0)
        nuevo_grupo = REPTIL;
    else
        nuevo_grupo = DESCONOCIDO;

    return nuevo_grupo;
}

// Funciones auxiliares
/**
 * Funcion pedir entero con recubrimiento de errores
 * No admite letras ni valores negativos
 */
int pedirEntero()
{
    // Declaración de variables
    int num = 0, esValido = 0;

    // Cuerpo funcion
    do
    {
        esValido = scanf("%d", &num);
        clearBuffer();
        if(esValido != 1 || num < 0)
        {
            printf(
                "\033[1;31mERROR: El tipo de dato introducido no es válido.\n"
                "-> \033[0m"
            );
            esValido = 0;
        }
    }
    while (!esValido);

    return num;
}

/**
 * Devuelve la cadena de caracteres que el usuario ha //escrito en el teclado por terminal
 * reservando memoria de manera dinamica
 */
char *leeLineaDinamica()
{
    // Declaración de variables
    char *destino = (char *)malloc(sizeof(char) * TAM_BLOQUE);
    int memoria_actual = TAM_BLOQUE;
    int i = 0;
    char c;

    // Cuerpo funcion
    while((c = getchar()) != '\n')
    {
        if(i == memoria_actual - 1)
        {
            memoria_actual += TAM_BLOQUE;
            destino = (char *)realloc(destino, sizeof(char) * memoria_actual);
        }
        destino[i] = c;
        i++;
    }
    destino[i] = '\0';

    return destino;
}

void clearBuffer()
{
    while(getchar() != '\n');
}

```