```c
/*
 PRACTICO 1
 Realizar una función que recibe un dato tipo char e imprime la representación decimal junto a sus representaciones binarias  en los formatos bit de signo, complemento a 1, complemento a 2 y binario desplazado (desde 127 a -127).
 */

void Imprimir(unsigned char num, int numBits)
{
    if (numBits > 1)
        Imprimir(num >> 1, numBits - 1);
    printf("%d", num & 1);
}

void Practico1(char numero)
{
    char copia=0;

    printf("Decimal: %4d\n",numero);

    printf("Bit de signo: ");
    if(numero>=0)
        Imprimir(numero, sizeof(char) * 8);
    else {
        copia=-numero;
        copia+=0x80;
        Imprimir(copia, sizeof(char) * 8);
        }
    printf("\n");

    printf("Complemento a 1: ");

    if(numero>=0)
        Imprimir(numero, sizeof(char) * 8);
    else {
        copia=-numero;
        copia=~copia;
        Imprimir(copia, sizeof(char) * 8);
        }
    printf("\n");

    printf("Complemento a 2: ");

    if(numero>=0)
        Imprimir(numero, sizeof(char) * 8);
    else {
        copia=-numero;
        copia=~copia;
        copia+=1;
        Imprimir(copia, sizeof(char) * 8);
        }
    printf("\n");

    printf("Binario desplazado: ");

    copia=0x7F+numero;
    Imprimir(copia, sizeof(char) * 8);
    printf("\n");

    return;
}

-----------------------------------------------------------------------
/*
```

```c
/*
PRACTICA 2
Realizar un programa que recibe por argumentos de man el nombre de un archivo de texto que contiene pares ordenados con la mediciones de un sensor. Cada par ordenado está formado por un valor de tiempo (t) y otro de amplitud (A) tal que en el archivo los valores del par ordenado están separados por coma. Luego imprimir:
    a) Las coordenadas de los máximos relativos.
    b) Las coordenadas de los mínimos relativos.
Usar estructuras para referenciar a los pares ordenados (t , A)
*/

#include <stdio.h>

// Estructura para almacenar los pares ordenados
struct ParOrdenado {
    int t;
    int A;
};

int main(int argc, char *argv[])
{
    struct ParOrdenado parAnterior, parActual, parSiguiente;
    int maxRelativoCount = 0, minRelativoCount = 0;
    FILE *archivo;

    if(argc!=2)
    {
        printf("Falta argumento con nombre del archivo\n");
        return -1;
    }
    archivo= fopen(argv[1], "r");

    if (archivo == NULL)
    {
        printf("No se pudo abrir el archivo.");
        return -2;
    }



    fscanf(archivo, "%d, %d", &parAnterior.t, &parAnterior.A);
    fscanf(archivo, "%d, %d", &parActual.t, &parActual.A);

    while (fscanf(archivo, "%d, %d", &parSiguiente.t, &parSiguiente.A) == 2)
    {
        // Comprueba si parActual.A es un máximo o mínimo relativo
        if (parAnterior.A < parActual.A && parActual.A > parSiguiente.A) {
            printf("Máximo relativo: (%d, %d)\n", parActual.t, parActual.A);
            maxRelativoCount++;
        } else if (parAnterior.A > parActual.A && parActual.A < parSiguiente.A) {
            printf("Mínimo relativo: (%d, %d)\n", parActual.t, parActual.A);
            minRelativoCount++;
        }

        // Actualiza los valores de los pares ordenados
        parAnterior = parActual;
        parActual = parSiguiente;
    }

    // Cierra el archivo
    fclose(archivo);

    // Imprime el número de máximos y mínimos relativos encontrados, no pedido en el enunciado.
    printf("\nNúmero de máximos relativos: %d\n", maxRelativoCount);
```

```c
    printf("Número de mínimos relativos: %d\n", minRelativoCount);

    return 0;
}

-----------------------------------------------------------------------
/*
 PRACTICA 3
 Hacer una función que recibe como argumentos el número de filas y columnas de una matriz, retornando un puntero a esa matriz; también realizar la función correspondiente a liberar la memoria utilizada para dicha matriz.
 */

// Implementación de la función crearMatriz
float **crearMatriz(int filas, int columnas)
{

    float **matriz = NULL, valor = 1.0;
    int i,j;

    // Reserva memoria para el arreglo de punteros a las filas
    matriz = (float **)malloc(filas * sizeof(float *));
    if (matriz == NULL)
    {
        printf("Error: No se pudo asignar memoria para la matriz.\n");
        return NULL;
    }

    // Reserva memoria para cada fila
    for (i= 0; i < filas; i++) {
        matriz[i] = (float *)malloc(columnas * sizeof(float));
        if (matriz[i] == NULL) {
            printf("Error: No se pudo asignar memoria para la fila %d.\n", i);
            liberarMatriz(matriz, i); // Libera la memoria asignada previamente
            return NULL;
        }
    }

    // Inicializa la matriz con valores de ejemplo
    for (i = 0; i < filas; i++)
    {
        for (j= 0; j < columnas; j++) {
            matriz[i][j] = valor;
            valor += 1.0;
        }
    }

    return matriz;
}

// Implementación de la función liberarMatriz
void liberarMatriz(float **matriz, int filas)
{
    int i;

    for (i = 0; i < filas; i++)
    {
        free(matriz[i]);
    }
    free(matriz);
    return;
}

```