Ingeniería Electrónica
Informática II
Procesamiento de archivos

Ing José Luis MARTÍNEZ

1. Procesamiento de archivos en C

El procesamiento de archivos es necesario para guardar datos en memoria. C ve a cada archivo como un secuencia de bytes, cada archivo finaliza en un EOF o ante un número específico de bytes determinado por el sistema.

Las siguientes instrucciones son las que se utilizan para el manejo de archivos:

1 **fopen().** Abre un fichero
Biblioteca: stdio.h
Declaración: *FILE* fopen (const char * nombre, const char*modo);*
Parámetros: El nombre del fichero y el modo de apertura
Valor devuelto: Un puntero a fichero (NULL en caso de error)
Detalles: Abre un fichero. Devuelve NULL si no se ha podido abrir correctamente
Ejemplo:
fichero=fopen("datos.txt", rt"); if (fichero == NULL) ...

2 **fclose().** Cierra un fichero
Biblioteca: stdio.h
Declaración: *int fclose (FILE* fichero);*
Parámetros: El identificador del fichero
Valor devuelto: 0 si todo va bien; EOF en caso de error
Detalles: Cierra un fichero. Si hay datos en buffer pendientes de volcar, se vuelcan antes de cerrar. Un fichero abierto con "fopen"se cerrará también automáticamente al terminar el programa. Si se intenta cerra un fichero no abierto, se obtendrá una "violación de segmento"
Ejemplo: *fclose(fichero);*

3 **fgetc().** Lee un carácter desde un fichero
Biblioteca: stdio.h
Declaración: *int fgetc (FILE* fichero);*
Parámetros: El identificador del fichero
Valor devuelto: El siguiente carácter en el fichero
Detalles: Devuelve el siguiente carácter del fichero, o EOF en caso de error
Ejemplo:
letra=fgetc(ficheroDatos);

4 **fgets().** Lee una cadena de texto desde fichero
Biblioteca: stdio.h
Declaración: *char * fgets (char * cadena, int tamaño, FILE * fichero);*
Parámetros: El identificador del fichero, la longitud máxima, la variable en la que se quiere guardar la

cadena
Valor devuelto: La cadena leída
Detalles:
Lee una cadena desde fichero (conserva el avance de línea del final).
Devuelve la cadena; en caso de error, devuelve NULL. Lee hasta encontrar un avance de línea o alcanzar la longitud máxima que se ha indicado

Ejemplo: *fgets(fichero, 30, nombre);*

5 **fputc().** Guarda un carácter en fichero
Biblioteca: stdio.h
Declaración: *int fputc (int c, FILE* fichero);*
Parámetros: El carácter a guardar, el identificador del fichero
Valor devuelto: El carácter escrito si todo ha ido bien; EOF en c
Ejemplo: *fputc(letra, ficheroSalida);*

6 **fputs().** Guarda una cadena de texto en fichero. No añade un avance de línea al final de la cadena (al contrario que "puts").
Biblioteca: stdio.h
Declaración: *int fputs (const char *cadena, FILE* fichero);*
Parámetros: El identificador del fichero, la cadena a guardar
Valor devuelto: Un número (EOF en caso de error)
Detalles: Devuelve un valor no negativo; en caso de error, devuelve EOF
Ejemplo: *fputs("Hola", fichero);*

7 **fscanf().** Lee datos formateados desde fichero
Biblioteca: stdio.h
Declaración: *int fscanf (FILE* fichero, const char * formato[, argumento1, argumento2...]);*
Parámetros: El identificador del fichero, la cadena de formato y las variables en que se guardarán los datos. Generalmente las variables deberán aparecer precedidas por "&"(excepto cadenas). Devuelve la cantidad de datos leídos (0 si ninguno, EOF en caso de error)
Valor devuelto: La cantidad de datos leídos
Detalles: Lee valores desde teclado, siguiendo un cierto código de formato
Ejemplo: *fscanf(fichero, "%d", &numero);*

8 **fprintf().** Guarda texto formateado en fichero
Biblioteca: stdio.h
Declaración: *int fprintf (FILE* fichero, const char * formato[, argumento1, argumento2...]);*
Parámetros: El identificador del fichero, la cadena de formato y los datos a guardar
Valor devuelto: La cantidad de letras escritas
Detalles: Escribe texto en un fichero siguiendo un cierto código de formato. Si no se indican suficientes argumentos para completar los códigos de formato, el resultado es indeterminado. Devuelve el número de caracteres escritos.
Ejemplo: *fprintf(fichero,"%s%d ", nombre, numero);*

9 **fread().** Lee datos desde fichero
Biblioteca: stdio.h
Declaración: *size_t fread(void *datos, size_t tamaño, size_t cantidad, FILE *fichero);*
Parámetros: El bloque de datos, el tamaño de cada dato, la cantidad de datos, el identificador del fichero
Valor devuelto: El número de datos leídos (en caso de error, ser

Detalles: Lee datos de cualquier tipo (cuyo tamaño se conozca) desde un fichero
Ejemplo: *fread(&ficha, sizeof(ficha), 1, fichero);*

10 **fwrite().** Guarda datos de cualquier tipo en un fichero
Biblioteca: stdio.h
Declaración: *size_t fwrite(void *datos, size_t tamaño, size_t cantidad, FILE *fichero);*
Parámetros: El bloque de datos, el tamaño de cada dato, la cantidad de datos, el identificador del fichero
Valor devuelto: El número de datos guardados .
Detalles: Devuelve un valor no negativo; en caso de error, devuelve EOF
Ejemplo: *fwrite(&arrayDeInt, sizeof(int), 100, fichero);*

11 **fseek().** Salta a una cierta posición de un fichero
Biblioteca: stdio.h
Declaración: *int fseek (FILE* fichero, long desplazamiento, int desde);*
Parámetros: El identificador del fichero, el desplazamiento, la posición de origen
Valor devuelto: Un número (0 si no ha habido problemas)
Detalles: Desplaza la posición actual de lectura/escritura del fichero a otro punto. El desplazamiento puede ser positivo (avanzar), cero o negativo (retroceder). La posición de origen se puede indicar con la ayuda de tres constantes:

- SEEK_SET (0, comienzo)
- SEEK_CUR (1, actual)
- SEEK_END (2, final)

Ejemplo: *fseek(fichero, -10, SEEK_CUR);*

12 **ftell().** Indica la posición actual en un fichero
Biblioteca: stdio.h
Declaración: *long ftell (FILE* fichero);*
Parámetros: El identificador del fichero
Valor devuelto: La posición actual
Detalles: Devuelve la posición actual en un fichero (-1 en caso de error)
Ejemplo: *posicionActual = ftell(ficheroDatos);*

13 **fgetpos().** Obtiene el valor actual de la posición del puntero dentro del archivo.
Biblioteca: stdio.h
Declaración: *int fgetpos(FILE * restrict stream, fpos_t * restrict pos);*
Parámetros: Coloca un valor *fpos_t* en la ubicación apuntada por *pos*, dentro del archivo. La palabra reservada *restrict* es del estándar C99, y se utiliza junto con un puntero para indicar que solamente con punteros se puede acceder al dato.
Valor devuelto: Devuelve 0 si la función es llevada con éxito, de lo contrario devuelve un valor distinto.
Detalles:las funciones *fseek()* y *ftell()* están limitadas por archivos cuyo tamaño puede ser representado por una variable *long*. *fgetpos()* elimina esta restricción, utilizando un nuevo tipo denominado *fpos_t (file position type)*, no es un tipo de variable fundamental, sino que es definido en términos de otros tipos de variables. Especifica una ubicación dentro de un archivo y no puede ser una variable de tipo array. En consecuencia la implementación es lo que define el tipo de la variable de acuerdo a las necesidades, por ejemplo puede ser una estructura.
Ejemplo: *fgetpos(fichero, &posicion);*

14 **fsetpos().** Biblioteca: stdio.h
Declaración: *int fsetpos(FILE *stream, const fpos_t *pos);* Valor devuelto: La función fsetpos retorna

cero si es llevada a cabo con éxito, si falla retorna un valor distinto a cero
Parámetros: Utiliza el valor *fpos_t* apuntado por *pos* para colocar el puntero en ese valor, dentro del *stream (archivo)*. El valor fpos_t debe obtenerse previamente mediante el llamado a *fgetpos()*.
Detalles: Descritos en *fgetpos()*
Ejemplo: *fsetpos(fichero, &comienzo);*

15 **rewind().** Coloca el indicador de posición de fichero al comienzo.
Biblioteca: stdio.h
Declaración: *void rewind(FILE *stream);*
Parámetros: Puntero al archivo.
Valor devuelto: No devuelve valor.
Detalles:
Ejemplo: *rewind(fichero);*

16 **feof().** Indica si se ha llegado al final de un fichero.
Biblioteca: stdio.h
Declaración: *int feof (FILE* fichero);*
Parámetros: El identificador del fichero
Valor devuelto: 0 (no alcanzado) o distinto de cero (sí alcanzado
Detalles: Devuelve 0 ("falso") si no se ha alcanzado el final del fichero, o un valor distinto de cero ("verdadero") si se ha llegado
Ejemplo: *while(!feof(fichero))* ...

17 **exit().** Termina la ejecución del programa
Biblioteca: stdlib.h
Declaración: *void exit (int estado);*
Parámetros: El código de error (o 0 si no hay error)
Valor devuelto: (Ninguno)
Detalles: Lo habitual es salir con el código 0 .exit(0)cuando todo ha funcionado correctamente, o con otro código en caso de error.
Ejemplo: *if (fichero==NULL) exit(1);*

18 **fflush()** Vaciado del buffer
Biblioteca: stdio.h
Declaración: *int fflush(FILE *stream);*
Parámetros: Puntero al archivo de salida
Valor devuelto:La función fflush retorna cero si el stream fue despejado con éxito. Si se detectaron errores, entonces retorna EOF.
Detalles: Si stream apunta a un stream de salida o de actualización cuya operación más reciente no era de entrada, la función fflush envía cualquier dato aún sin escribir al entorno local o a ser escrito en el fichero; si no, entonces el comportamiento no está definido. Si stream es un puntero nulo, la función fflush realiza el despeje para todos los streams cuyo comportamiento está descrito anteriormente.
Ejemplo: *fflush(stdout);*

19 **ferror()** Comprueba el indicador de errores Biblioteca: stdio.h
Declaración: *int ferror(FILE *stream);*
Parámetros: Puntero al archivo de entrada/salida
Valor devuelto:La función ferror retorna un valor distinto a cero si y sólo si el indicador de errores está activado para stream..

Detalles: Cuando una función de entrada estándar devuelve EOF generalmente significa que encontró el final del archivo pero también puede indicar que ocurrió un error. Las funciones *feof()* y *ferror()* ayudan a distinguir las dos posibilidades. *feof()* devuelve un valor distinto de cero si detectó el EOF. *ferror()* devuelve un valor distinto de cero, si se produjo un error de lectura o escritura. Ejemplo: *if(ferror(fichero))*

Cuando se abre un archivo se le asocia un flujo que puede ser:

- *standard input (stdin)* Normalmente el teclado pero puede ser un archivo
- *standard output (stdout)* Normalmente la pantalla, también se la puede enviar a un archivo.
- *standard error (stderr)* Normalmente la pantalla o también un archivo

Al abrir un archivo se devuelve un puntero a la estructura *FILE definida en stdio.h.

```
#ifndef _FILE_DEFINED
#define _FILE_DEFINED
typedef struct _iobuf
{
char* _ptr;
int _cnt;
char* _base;
int _flag;
int _file;
int _charbuf;
int _bufsiz;
char* _tmpfname;
} FILE;
#endif /* Not _FILE_DEFINED */
```

Un archivo puede ser abierto de varias formas, como se muestra a continuación

Modo	**Significado**
(read) "r"	Abrir un archivo de texto para lectura
(write) "w"	Abrir un archivo de texto para escritura,si el archivo no existe lo crea, en cambio si existe su longitud es truncada a cero perdiendo los datos.
(append) "a"	Abre un archivo de texto para escritura agregando contenido al final del archivo, si este no existe lo crea.
"$r+$"	Abre un archivo de texto para actualizarlo , es decir hace lectura y escritura
"$w+$"	Abre un archivo de texto para actualización (lectura y escritura), primero eliminando el contenido si existe el archivo, si no existe lo crea.
"$a+$"	Abre un archivo de texto para actualización (lectura o escritura), agregando el nuevo contenido al final o creando el archivo si no existe. Todo el archivo puede ser leído, pero solo la escritura puede ser agregada.
"rb", "wb", "ab", "$ab+$", "$a+b$", "$wb+$", "$w+b$", "$ab+$", "$a+b$"	Misma función que los modos descritos anteriormente pero se utiliza para archivos binarios en lugar de archivos de texto.

Ejemplo. Leer un archivo de texto y mostrarlo en la salida estándar (pantalla), si encuentra un ';' saltar de línea.

```
\\ Archivo de Texto

Archivo de prueba,
para; obtener
datos
```

El programa será el siguiente:

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

int main()
{
    char p1;
    FILE *ptrArchivo;  // puntero a la estructura FILE
    ptrArchivo = fopen("ArchivoDeTexto.txt","r"); // abro el archivo para lectura

    if(ptrArchivo != NULL)  // verifico que el archivo existe
        {
            while(!feof(ptrArchivo))
            {
                //fscanf(ptrArchivo, "%s", &p1);
                if((p1 =fgetc(ptrArchivo)) == ';') // me fijo si me topo con un ;
                                        // se recorre el archivo hasta el EOF con fgetc
                {                                   // obteniendo un caracter a la vez
                    printf("\n");
                }
                else
                    putchar(p1);
            }
        }
    fclose(ptrArchivo);  // cierro el archivo
    cin.get();

    return 0;//EXIT_SUCCESS;
}
```

Ejemplo. Realice un programa en C que tome como argumento de entrada en archivo .txt y cuente la cantidad de caracteres que tiene. El programa debe ser general para leer cualquier archivo.

```
/* contarChar.c -- Cuenta la cantidad de letras de un archivo*/
#include <stdio.h>
#include <stdlib.h> //
int main(int argc, char *argv[])
{
 int ch; // variable que lee cada caracter
 FILE *fp; // "file pointer"
 long count = 0;
```

```
 if (argc != 2)
    {
        printf("Ha olvidado el archivo que tiene que leer con: %s \n", argv[0]);
      // coloca el nombre del ejecutable
      // al ser distinto de 2 es porque
      // no cargo el fichero que debe leer
        exit(1);
    }
    if ((fp = fopen(argv[1], "r")) == NULL)
        {
            printf("No se puede abrir %s\n", argv[1]);
            exit(1);
            //exit(EXIT_FAILURE);
        }

 while ((ch = getc(fp)) != EOF)
    {
        putc(ch,stdout); // igual a putchar(ch);
        count++;
    }
 fclose(fp);
 printf("\n\n\t\tEl archivo %s tiene %ld caracteres\n", argv[1], count);
 return EXIT_SUCCESS;
}
```

Ejemplo. Escribir un programa que calcule la raíz cuadrada de un número entero, que tenga la siguiente interacción con el usuario:

a Solicitar al usuario un número entero (cero para salir)

b Mostrar en la salida estándar (stdout) la raíz cuadrada del entero ingresado

c Mostrar en la salida de error estándar (stderr) si el entero es negativo. Ejecute el programa

   a Redireccionando a un archivo la salida estándar

   b Redireccionando a un archivo la salida de error est´andar

   c Redireccionando la entrada desde un archivo

   d Redireccionando la salida de error estándar al archivo /dev/null (Utilizar la función fprintf() de biblioteca de entrada/salida estándar (stdio). Ver la página de manual de dicha función.)

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main()
{
int num;
float res;
FILE *fp, *err;

if((fp=fopen("salida.dat", "w")) == NULL)
fprintf(stdout, "El archivo no pudo abrirse.\n");
else
fprintf(stdout, "El archivo se creo con exito para escritura\n");
```

```
if((err=fopen("error.dat", "w")) == NULL)
fprintf(stdout, "El archivo de errores no pudo abrirse.\n");
else
fprintf(stdout, "El archivo de errores se creo con exito para escritura\n");
printf("Ingrese un numero, 0 para terminar. \n");
scanf("%d", &num);
while(num!=0)
{
if (num>0)
{
res = sqrt(num);
fprintf(fp, "%f\n",res);
stdout > "salida.dat";
}
else
fprintf(err, "No valen numeros negativos %d.\n", num);

printf("Ingrese un nuevo numero.\n");
scanf("%d", &num);


}
fclose(fp);
fclose(err);

return 0;
}
```

Ejemplo. Escriba un programa en C que tome un archivo de texto y guarde una copia del original pero tomando solamente un caracter de cada tres

```
// reducto.c -- reduce los archivos en dos tercios
#include <stdio.h>
#include <stdlib.h> // para exit()
#include <string.h> // para strcpy(), strcat()
#define LEN 40

int main(int argc, char *argv[])
{
    FILE *in, *out; // declara dos punteros FILE
    int ch;
    char name[LEN]; // array para el nombre del archivo de salida
    int count = 0;
    // verificar los argumento del comando de linea
    if (argc < 2)
        {
            fprintf(stderr, "Usar: %s nombre de archivo\n", argv[0]);
            exit(1);
        }
    // set up input
    if ((in = fopen(argv[1], "r")) == NULL)
        {
            fprintf(stderr, "No pude abrir el archivo \"%s\"\n", argv[1]);
```

```
            exit(2);
        }
    // set up output
    strncpy(name,argv[1], LEN - 5); // copiar el nombre del archivo a name
    name[LEN - 5] = ' \0';
    strcat(name,".red"); // append .red al archivo de salida, quedara "nombreArchivo.txt.red"
    if ((out = fopen(name, "w")) == NULL)
        { // abrir el archivo para escritura
            fprintf(stderr,"No se pudo crear el archivo de escritura. \n");
            exit(3);
        }
    // copy data
    while ((ch = getc(in)) != EOF)
        if (count++ % 3 == 0)
            putc(ch, out); // guarda cada 3 caracteres
    // clean up
    if (fclose(in) != 0 || fclose(out) != 0)
        fprintf(stderr,"Error al cerrar los archivos abiertos\n");
    return 0;
}
```

Ejemplo. Cree un archivo al que se le pueda agregar texto

```
/* agregaPalabras.c -- uso de fprintf(), fscanf(), y rewind() */
#include <stdio.h>
#include <stdlib.h>
#define MAX 40
int main(void)
{
    FILE *fp;
    char palabras[MAX];
    if ((fp = fopen("palabras", "a+")) == NULL)
        {
            fprintf(stdout,"No se puedo abrir el archivo \"palabras\". \n");
        exit(1);
        }
    puts("Ingrese las palabras a agregar al archivo, presione la tecla Enter");
    puts(" al principio de la linea para terminar.");
    while (gets(palabras) != NULL && palabras[0] != ' \0')
        fprintf(fp, "%s ", palabras);
    puts("Contenido del archivo palabras:");
    rewind(fp); /* regresar al principio del archivo */
    while (fscanf(fp,"%s",palabras) == 1)
        puts(palabras);
    if (fclose(fp) != 0)
        fprintf(stderr,"Error al cerrar el archivo\n");
    return 0;
}
```

Ejemplo. Escriba un programa que repita frases como un loro

```
/* loro.c -- uso de fgets() y fputs() */
```

```
#include <stdio.h>
#define MAXLINE 20
int main(void)
{
    char line[MAXLINE];
    while (fgets(line, MAXLINE, stdin) != NULL &&  line[0] != ' \n')
        fputs(line, stdout);
    return 0;
}
```

Ejemplo. Escriba un programa que tome un archivo de texto y lo muestre en forma inversa

```
/* revertir.c -- muestra el texto de un archivo en orden inverso */
#include <stdio.h>
#include <stdlib.h>
#define CNTL_Z ' \032' /* eof */
#define SLEN 50
int main(void)
{
    char file[SLEN];
    char ch;
    FILE *fp;
    long count, last;
    puts("Ingrese el nombre del archivo que quiere procesar:");
    gets(file);
    if ((fp = fopen(file,"rb")) == NULL)
        { /* lectura en modo binario*/
            printf("reverse can't open %s\n", file);
            exit(1);
        }
    fseek(fp, 0L, SEEK_END); /* ir al final del archivo */
    last = ftell(fp);  // Obtiene la posición del puntero fp que apunta al final
    for (count = 1L; count <= last; count++)
        {
            fseek(fp, -count, SEEK_END); /* retroceder una posicion */
            ch = getc(fp);
            if (ch != CNTL_Z && ch != ' \r')
                putchar(ch);
        }
    putchar(' \n') ;
    fclose(fp);
    return 0;
}
```

¿Como funcionan *fseek()* y *ftell()*? El primero de los tres argumentos de *fseek()* es un puntero FILE al archivo en el que se debe buscar, este archivo debe estar previamente abierto con *fopen()*. El segundo argumento de *fseek()* es el offset, indica cuanto hay que moverse desde el inicio del archivo, el valor puede ser positivo (adelanta), negativo (atrasa) o cero (queda en el lugar). El tercer argumento indica el modo que puede ser:

- SEEK_SET (0, comienzo)
- SEEK_CUR (1, actual)

- SEEK_END (2, final)

Algunos ejemplos:

```
fseek(fp, 0L, SEEK_SET); // ir al inicio del archivo
fseek(fp, 10L, SEEK_SET); // ir 10 bytes a partir del inicio
fseek(fp, 2L, SEEK_CUR); // avance 2 bytes desde la posicion actual
fseek(fp, 0L, SEEK_END); // ir al final del archivo
fseek(fp, -10L, SEEK_END); // retroceder 10 bytes desde el final del archivo
```

1.1. Acceso secuencial vs acceso aleatorio

Los archivos se crean y almacenan en forma secuencial, hasta ahora hemos visto texto plano, pero si queremos guardar datos entonces puede ser necesario modificarlos, por ejemplo obtengo los valores de un ensayo de laboratorio replicado por varios técnicos, si deseo cambiar el nombre de alguno de los técnicos o los valores, puede que la longitud de los nuevos datos pisen la información de los datos ya guardados, para evitar estos inconvenientes abrimos los archivos en modo binario y reservamos bloques de memoria. Con los archivos binarios puedo utilizar las funciones de *fseek()*

Ejemplo. Crear un archivo de acceso secuencial

```
/*creaSecuencial.c crea un archivo de acceso secuencial*/
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int cuenta;
    char nombre[30];
    double saldo;
    FILE *ptrCf;

    if (( ptrCf=fopen("clientes.dat", "a+")) == NULL)
        {
            puts("No se pudo crear o abrir el archivo");
            exit(EXIT_FAILURE);
        }
    else
        {
            puts("Ingrese el numero de cuenta, nombre y saldo.");
            puts("Ingrese EOF para finalizar la carga.");
            printf("%s", "# ");

            scanf("%d%29s%lf", &cuenta, nombre, &saldo);
            // Escribir en el archivo la cuenta, nombre y saldo con fprintf
            while (!feof(stdin) )
                {
                    fprintf(ptrCf, "%d %s %.2f\n", cuenta, nombre, saldo);
                    printf("%s", "# ");
                    scanf("%d%29s%lf", &cuenta, nombre, &saldo);
                }

        }
    rewind(ptrCf);
```

```
    printf("\n\n Cuenta\t nombre\t balance ");
    while(!feof(ptrCf))
    {
        fscanf(ptrCf, "%d%s%lf", &cuenta, nombre, &saldo);
        printf("\n%d\t %s\t %.2f", cuenta, nombre, saldo);
    }
    fclose(ptrCf);
    return EXIT_SUCCESS;

}
```

Ejemplo. Crear un archivo de acceso aleatorio

```
/*creaAleatorio.c archivo de acceso aleatorio*/

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

struct datosCliente
{
    int numCta;
    char apellido[15];
    char nombre[10];
    double saldo;
};

int main()
{
    FILE *ptrCf;
    struct datosCliente cliente={0, "", "", 0.0};

    if((ptrCf=fopen("credito.dat", "wb")) == NULL)
    {
        printf("\nEl archivo no pudo abrirse o crearse");
        exit(EXIT_FAILURE);
    }
    else
    {
        printf("\nIngrese el numero de cuenta, 0 para terminar la entrada.\n ");
        scanf("%d", &cliente.numCta);
        while(cliente.numCta != 0)
        {
            printf("\n Introduzca el apellido, nombre y saldo. \n");
            fscanf(stdin, "%s%s%lf", cliente.apellido, cliente.nombre, &cliente.saldo);
            fseek(ptrCf, (cliente.numCta-1) * sizeof(struct datosCliente), SEEK_SET);
            fwrite(&cliente, sizeof(struct datosCliente), 1, ptrCf);
            printf("\nIngrese el numero de cuenta, 0 para terminar la entrada.\n ");
            scanf("%d", &cliente.numCta);
        }
        fclose(ptrCf);

    }
```

```
    return EXIT_SUCCESS;
}
```

Problemas

1. ¿Qué errores tiene este programa?

   ```
   int main(void)
   {
    int * fp;
    int k;
    fp = fopen("gelatina");
    for (k = 0; k < 30; k++)
    fputs(fp, "Marcvia come gelatina.");
    fclose("gelatina");
    return 0;
   }
   ```

2. ¿Qué hace el siguiente programa?

   ```
   #include <stdio.h>
   #include <stdlib.h>
   #include <ctype.h>
   int main(int argc, char *argv[])
   {
    int ch;
    FILE *fp;
    if (argc < 2)
    exit(2);
    if ( (fp = fopen(argv[1], "r")) == NULL)
       exit(1);
    while ( (ch= getc(fp)) != EOF )
       if( isdigit(ch) )
           putchar(ch);
    fclose (fp);
    return 0;
   }
   ```

3. Suponga que tiene estas declaraciones en el programa

   ```
   #include <stdio.h>
   FILE * fp1,* fp2;
   char ch;
   fp1 = fopen("terky", "r");
   fp2 = fopen("jerky", "w");
   ```

   Suponga también que ambos se abrieron sin problemas. Complete los argumentos

   ```
   a. ch = getc();
   b. fprintf(,"%c\n", );
   c. putc(, );
   d. fclose(); /* close the terky file */
   ```

4. Escriba un programa que no toma argumentos de la línea de comandos, pero si llega a haber un argumento entonces este es tomado como el nombre del del archivo, caso contrario la entrada estándar (stdin) es utilizada para ingresar los datos. Los datos son números en punto flotante, calcule la media aritmética de los números ingresados.
   NOTA: Los datos ingresados deben ser guardados en un archivo junto con la media calculada.

5. DUP que tome dos argumentos de la línea de comandos, el primero es un caracter y el segundo es el nombre del archivo. El programa debe imprimir solamente los renglones que tengan el caracter ingresado.

6. Investigue cuál es la diferencia:

   a) Entre guardar 8238201 usando fprintf() y usando fwrite()

   b) Entre guardar el caracter S usando putc() y usando fwrite()

7. gets() es una función peligrosa ¿Porqué?

8. ¿En qué se diferencian?

```
printf("Hello, %s\n", name);
fprintf(stdout, "Hello, %s\n", name);
fprintf(stderr, "Hello, %s\n", name);
```

9. Escriba un programa que tome dos archivos, debe tener la flexibilidad de que los archivos sean leídos de la línea de comandos o solicitados al usuario que ingrese el los nombres

   a) Imprima la primera línea del archivo 1, a continuación en el siguiente renglón la primera línea del archivo 2, intercalando así todas las líneas

   b) Modifique el programa para que el intercalado sea realizado en el mismo renglón