Archivos

Introducción

Todos los programas que se ejecutaron hasta el momento requerían de ingreso de datos para realizar determinadas tareas y luego mostrar resultados. Tales programas tienen un gran defecto y es que los datos que se cargan desaparecen cuando el programa termina , por lo tanto cada vez que se corra el programa hay que cargar los datos nuevamente.

Esto ocurre debido a que los datos ingresados quedan guardados en memoria y al terminar el programa todos los espacios de memoria reservados son devueltos al sistema operativo perdiendo todos los datos que se habían cargado.

Para solucionar el problema debería existir una forma de almacenar los datos en forma permanente, por ejemplo en un medio magnético como el disco rígido de tal forma de disponer de la información en cualquier momento.

Concepto de Archivo

Un *archivo de datos* es un conjunto de registros relacionados, que se trata como una unidad y se almacena sobre un dispositivo de almacenamiento externo.

Es un conjunto de datos estructurados en una colección de entidades elementales o básicas denominadas registros que son de igual tipo y constan a su vez de diferentes entidades de nivel más bajos denominadas campos.

Tipos de Archivos- Según Método de Acceso

En función del método de acceso a los datos, que depende del modo físico en el que se ha organizado su información:

- **Ficheros de Acceso Secuencial:** Se accederá a cada elemento del archivo uno tras otro, en el mismo orden en el que se situaron, es decir para llegar a un registro se recorren todos los precedentes.

- **Ficheros de Acceso Directo:** Permiten el posicionamiento directo sobre un determinado registro mediante la especificación de un índice, que determina la posición del registro repecto al origen del archivo.

Tipos de Archivos- Según tipo de elementos

Esta clasificación hace referencia a la forma en la cuál se interpretan los datos, se debe tener en claro que la única forma de guardar datos en el disco es en formato binario.

Todos los archivos tienen al final una marca que indica el final del archivo. Dicha marca se conoce como EOF(End Of File).

En función del tipo de elementos a almacenar, existen 2 tipos de archivos que son :

- Archivos de texto
- Archivos binarios

Tipos de Archivos- Según tipo de elementos

Archivos de Texto

Un archivo de texto es una secuencia de caracteres.

- **Un archivo de texto contiene toda su información guardada en binario pero se interpreta como texto.**

- **Absolutamente todo lo que contiene debe ser interpretado como texto , ya que cuando se escribe el archivo , los datos son enviados como caracteres.**

- **Los archivos de texto se caracterizan por ser planos, es decir, todas las letras tienen el mismo formato y no hay palabras subrayadas, en negrita, o letras de distinto tamaño o ancho.**

Tipos de Archivos- Según tipo de elementos

Archivos Binarios

En un archivo binario se guardan datos con distinto formato , es decir se pueden guardar caracteres mezclados con enteros y flotantes.

El hecho de trabajar con archivos binarios esta asociado con el uso de estructuras , dado que la forma mas simple de trabajar es cargar una estructura para luego escribirla en el archivo y de esa forma guardar los datos.

Apertura de un Archivo

El *modo es una cadena de caracteres que indica el tipo del fichero (texto o* binario) y el uso que se va ha hacer de él lectura, escritura, añadir datos al final, etc. La siguiente tabla muestra los valores permitidos para modo.

Modo	Significado
r	Abre un archivo de texto para lectura.
w	Crea un archivo de texto para escritura.
a	Abre un archivo de texto para añadir.
rb	Abre un archivo binario para lectura.
wb	Crea un archivo binario para escritura.
ab	Abre un archivo binario para añadir.
r+	Abre un archivo de texto para lectura / escritura.
w+	Crea un archivo de texto para lectura / escritura.
a+	Añade o crea un archivo de texto para lectura / escritura.
r+b	Abre un archivo binario para lectura / escritura.
w+b	Crea un archivo binario para lectura / escritura.
a+b	Añade o crea un archivo binario para lectura / escritura.

Apertura de un Archivo

En cuanto a los valores permitidos para los bytes, se puede añadir otro carácter a la cadena de modo:

- t: modo texto. Normalmente es el modo por defecto. Se suele omitir.
- b: modo binario.

Funciones de Entrada Salida

En C , todas las operaciones que se realizan sobre archivos son hechas a través de funciones. Básicamente existen 2 categorías de funciones para trabajar con archivos y son las que usan “buffer” y las que acceden directamente al archivo.

El hecho de utilizar un buffer significa que no se tiene acceso directo al archivo y que cualquier operación que se desee realizar (lectura o escritura) va a ser hecha sobre el buffer. Cuando el buffer se llena o se vacía se actualizan los datos desde y hacia el archivo.

Se debe tener en cuenta que al usar las funciones sobre archivos estamos trabajando con un intermediario que accede al archivo. Por lo tanto todos los datos que van al archivo (escritura) y que vienen (lectura) se encuentran en memoria y fread y fwrite toman o dejan esos datos en el archivo.

Funciones de Entrada Salida

Se puede conseguir la entrada y la salida de datos a un archivo a través del uso de la biblioteca de funciones.

C no tiene palabras claves que realicen las operaciones de E/S. La siguiente tabla da un breve resumen de las funciones que se pueden utilizar. Se debe incluir la librería STDIO.H.

Nombre	Función
fopen()	Abre un archivo.
fclose()	Cierra un archivo.
fgets()	Lee una cadena de un archivo.
fputs()	Escribe una cadena en un archivo
fseek()	Busca un byte especifico de un archivo.
fprintf()	Escribe una salida con formato en el archivo.
fscanf()	Lee una entrada con formato desde el archivo.
feof()	Devuelve cierto si se llega al final del archivo.
ferror()	Devuelve cierto si se produce un error.
rewind()	Coloca el localizador de posición del archivo al principio del mismo.
remove()	Borra un archivo.
fflush()	Vacía un archivo.

Puntero a un Archivo

Un puntero a un archivo es un puntero a una información que define el nombre, el estado y la posición actual del archivo. En esencia identifica un archivo especifico y utiliza la secuencia asociada para dirigir el funcionamiento de las funciones de E/S con buffer.

Un puntero a un archivo es una variable de tipo puntero al tipo FILE que se define en STDIO.H. Un programa necesita utilizar punteros a archivos para leer o escribir en los mismos. Para obtener una variable de este tipo se utiliza una secuencia como esta:

```
FILE *NombreArchivo
```

Ejemplo: `FILE *arch;`

Punteros

Los punteros en el Lenguaje C , son variables que " apuntan " , es decir que poseen la dirección de las ubicaciones en memoria de otras variables, y por medio de ellos se tiene un método de acceso a todas ellas .

Los punteros se declaran utilizando el operador * (asterisco) entre el tipo y el identificador de la variable, por ejemplo:

```
char *ptr;
```

Con el operador *, se declara una variable como puntero.

Los punteros son variables que almacenen números enteros. El tipo char indica que en “ptr” se almacenara una dirección de una variable que es de tipo char.

Apertura de un Archivo – fopen()

La función fopen() abre una secuencia para que pueda ser utilizada y la asocia a un archivo. Su prototipo es:

FILE *fopen(const char nombre_archivo, cost charmodo);

fichero = fopen (nombre-fichero, modo);

Donde nombre_archivo es un puntero a una cadena de caracteres que representan un nombre valido del archivo y puede incluir una especificación del directorio. La cadena a la que apunta modo determina como se abre el archivo.

Apertura de un Archivo – fopen() - EJEMPLO

La forma habitual de utilizar la instrucción fopen es dentro de una sentencia condicional que permita conocer si se ha producido o no error en la apertura, por ejemplo:

```
FILE *fich;

if ((fich = fopen("nomfich.dat", "r")) == NULL)
{ /* control del error de apertura * /

printf ( " Error en la apertura. Es posible que el fichero no exista \n ");
}
```

El resultado de fopen se almacena en la variable fich y después se compara fich con NULL para saber si se ha producido algún error.

Apertura de un Archivo-EJEMPLO

FILE * datosdatos = fopen (“nombres.dat”,”r”);

Otras formas de abrir el archivo:

datos = fopen (“nombres.dat”, “w”);
datos = fopen (“nombres.dat”, “a”);
datos = fopen (“nombres.dat”, “ra”);

¿Por qué no se puede abrir un fichero?

- **No existe el archivo**
- **Disco lleno**
- **Nombre incorrecto**
- **Directorio no válido**

Apertura de un Archivo-EJEMPLO

```
……….
int main ()
{
   FILE *parch;
   if((parch=fopen(“c:\banco.dat”,”rb”))==NULL) //Se abre en modo  lectura
                if((parch=fopen(“c:\banco.dat”,”wb”))==NULL)
                    //Si el modo anterior dio error el archivo
                   {  //no existe , por lo tanto se crea
                        printf(“\nEl archivo no puede ser abierto”);
                   }
           fclose(parch);
Return 0;
}
```

La idea es abrir un archivo para leer , en el caso de que exista se trabaja normalmente , pero si no existe lo abre el segundo fopen. De esta forma nos evitamos borrar un archivo que existe y tiene datos.

Cierre de un Archivo – fclose()

La función fclose() cierra una secuencia que fue abierta mediante una llamada a fopen(). Escribe toda la información que todavía se encuentre en el buffer en el disco y realiza un cierre formal del archivo a nivel del sistema operativo.

Un error en el cierre de una secuencia puede generar todo tipo de problemas, incluyendo la pérdida de datos, destrucción de archivos y posibles errores intermitentes en el programa.

Todo archivo que se abre debe ser cerrado antes de terminar el programa.

Cierre de un Archivo – fclose()

El prototipo de esta función es:

```
int  fclose(FILE *F);
```

Donde F es el puntero al archivo devuelto por la llamada a fclose(). Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito.

Ejemplo

```
Int main ()
{
        FILE *parch;
        if((parch=fopen("c:\\banco.dat","rb"))==NULL)//Se abre en modo lectura
        printf("\nEl archivo no puede ser abierto");

        if((fclose(parch))= = -1) //Se cierra el archivo
                printf("\nNo se pudo cerrar el archivo");
        else
                printf("\nEl archivo se cerro exitosamente");
return 0;
}
```

Lectura y Escritura en un Archivo

Una vez que el archivo se encuentra abierto se puede empezar a trabajar para leer o escribir.

Para almacenar datos en un fichero es necesario realizar una operación de escritura, de igual forma que para obtener datos hay que efectuar una operación de lectura.

En C existen muchas y variadas operaciones para leer y escribir en un archivo. Se pude mencionar:

- **fread -fwrite,**
- **fgetc -fputc,**
- **fgets - fputs,**
- **fscanf –fprintf**

Es aconsejable utilizarlas por parejas; es decir, si se escribe con fwrite se debe leer con fread.

Lectura y Escritura en un Archivo

fread -fwrite

Para leer y escribir en ficheros que no sean de texto las operaciones que se deben utilizar son fread y fwrite.

El prototipo de la función de escritura es el siguiente:

int ***fwrite*** (void * *origen* , size_t *tamaño* , size_t *cantidad* , FILE **arch*)

Donde:

origen	Es un puntero al lugar desde donde se obtienen los datos para escribir en el archivo
tamaño	Es el tamaño en bytes del dato que se va a escribir
Cantidad	Es la cantidad de datos de longitud *tamaño* que se van a escribir
arch	Es el puntero a FILE asociado al archivo

Lectura y Escritura en un Archivo
fread -fwrite

Valor retornado: Devuelve el número de datos escritos (*cantidad*). Si el valor retornado es menor al que se indicó por medio de la variable *cantidad* , significa que hubo un error en la escritura.

La función *fwrite* toma *cantidad* de datos de longitud *tamaño* desde la dirección *origen* y los escribe en el archivo asociado al puntero *arch* comenzando desde la posición actual del indicador de posición del archivo.

Una vez que se completó la operación de escritura el indicador de posición es actualizado (queda apuntando al lugar donde se puede escribir el próximo dato).

fwrite - Ejemplo

```
……….
int main ()
{
    FILE *parch;
    char texto[ ]=”Prueba de escritura”;
    int cant , longi;
    if((parch=fopen(“c:\\prueba.txt”,”w”))==NULL)//Se abre en modo escritura
            printf(“\nEl archivo no puede ser abierto”);

longi=strlen (texto );
cant=fwrite ( texto , sizeof ( char ) , longi , parch );  //Se escribe al archivo

if (cant<longi)
            printf(“\nError al escribir el archivo”);
else
{
            printf(“\nSe escribieron %d caracteres”, cant);
            fclose(parch);
}
……
```

fwrite – Corregir el siguiente programa

```
#include <stdio.h>
#include <conio.h>
#define ARCH  "c:\\bin.dat"

struct a{
                char nombre[10];
                int edad;
        };

int main ()
{
        FILE  *bin;
        struct a pers;
        clrscr();
        printf("\nSe va a generar por primera vez el archivo\n");
        if ((bin=fopen(ARCH,"wb"))==NULL)
                printf("El archivo no puede ser abierto");
```

```
        do
          {
                    printf("\nIngrese el nombre: ");
                    gets(pers.nombre);
                    printf("Ingrese la edad: ");
                    scanf("%d",&pers.edad);
                    fwrite(&pers,sizeof(pers),1,bin);
                    printf("\nPresione 1 para terminar");

           }while((getche())!=1);

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

Antes de escribir el archivo se debe cargar el dato que se desea guardar , en nuestro caso debemos cargar la estructura. Una vez que se cargaron todos los campos se llama a fwrite y se le pasa la dirección de comienzo de la estructura (&pers) , el tamaño en bytes de la estructura (se puede escribir directamente o usar sizeof) , la cantidad de estructuras que se van a escribir (se cargó solo una por lo tanto se escribe una) y finalmente el puntero que hace referencia al archivo.

En el caso que no se desee escribir la estructura entera se deberán hacer 2 *fwrite* , uno para la edad y el otro para el nombre.

Lectura y Escritura en un Archivo
fread –Lectura de un archivo

Para realizar la lectura de un archivo se utiliza la función *fread* que tiene el siguiente prototipo

int *fread* (void * *destino* , size_t *tamaño* , size_t *cantidad* , FILE **arch*)

Donde:

destino	**Es un puntero al lugar donde se va a dejar el dato leido con fread**
tamaño	**Es el tamaño en bytes del dato a leer**
Cantidad	**Es la cantidad de elementos de longitud *tamaño* que se van a leer**
arch	**Es el puntero a la estructura FILE asociada al archivo desde el que se va a leer.**

Lectura y Escritura en un Archivo
fread –fwrite - Ejemplo

Valor retornado: : Devuelve el número de datos leidos (*cantidad*). Si el valor retornado es menor al que se indicó por medio de la variable *cantidad* , significa que hubo un error en la lectura o que se llegó al final de archivo.

La función *fread* lee desde el archivo referenciado por *arch* a partir de la posición actual del indicador de posición , *cantidad* de elementos de longitud *tamaño* y deja los elementos leidos en la dirección de memoria indicada por *destino*.

Una vez que se completó la operación de lectura se actualiza automáticamente el indicador de posición del archivo.

A diferencia de lo que ocurre en la escritura , se debe verificar que se realice la lectura mientras no se haya llegado al final del archivo. Esta operación se realiza por medio de la función *feof.*

Lectura y Escritura en un Archivo fread - Ejemplo

```
#include <stdio.h>
#include <conio.h>
#define ARCH      "c:\\bin.dat"

struct a{
                        char nombre[10];
                        int edad;
            };

int main ()
{
            FILE  *bin;
            struct a pers;
            int cant;
            clrscr();
            if ((bin=fopen(ARCH,"rb"))==NULL)
                        printf("No se pudo abrir el archivo");
```

```
while(!feof(bin))
{
                cant=fread(&pers,sizeof(pers),1,bin);
                if(cant!=1)
                {
                        if(feof(bin))
                                break;
                        else
                        {
                                error("No leyo el ultimo registro");
                                break;
                        }
                }
                printf("\n%s\t%d",pers.nombre,pers.edad);
}
fclose(bin);
getch();
return 0;
}
```

Después de hacer la lectura se debe verificar que se haya leido la cantidad de datos que se indicó. Ocurre que cuando no se lee la cantidad de datos indicada puede haberse alcanzado el final de archivo o se pudo haber producido un error.

Es por esto que cuando se entra al if que verifica la cantidad , se debe preguntar si se llego al final del archivo.

Lectura y Escritura formateada de texto – fprintf y fscanf

Sus prototipos son:

```
int fprintf(FILE *F, const char *cadena_de_control, .....);
int fscanf(FILE *F, const char *cadena_de_control, .....);
```

Donde F es un puntero al archivo devuelto por una llamada a fopen(). fprintf() y fscanf() dirigen sus operaciones de E/S al archivo al que apunta F.

Estas funciones se comportan exactamente como prinft() y scanf() discutidas anteriormente, excepto que operan sobre archivo.

Lectura y Escritura formateada de texto – fprintf y fscanf

Para que lo escrito con fprintf pueda ser correctamente leído con fscanf es conveniente que el formato en el que se indican los tipos de datos que se van a leer o escribir sean similares para ambas instrucciones, que los tipos de datos estén separados, por ejemplo, por un blanco y que tengan un fin de línea al final.

Ejemplo:

```
int num;
char car, cad [10] ;
FILE *f.
/* apertura del fichero */
.....
.....
fscanf (f, "%d %c %s ", &num, &car, cad);
fprintf ( f, "%d %c %s \n ", num, car, cad);
fscanf (f, "%s %c %d ", cad, &car, &num);
fprintf (f, "%s %c %d \n ", cad, car, num);
```

Ejemplo

El archivo incluirá los datos de la siguiente forma:

Nombre del alumno1 nota
Nombre del alumno2 nota
.....

Algunas veces se necesita manipular por separado el nombre del alumno y su nota, para esto es necesario separarlo en campos. Se puede realizar introduciendo caracteres delimitadores entre campo y campo, por ejemplo:

```
fprinft(C,"%s;%d \n",nombre,cal);
```

Esto generara un archivo de tipo:

Nombre del alumno1;nota
Nombre del alumno2;nota
.....

Ejemplo:

```
 #include<stdio.h>

int  main();
{
FILE *fich;
int edad;

fich = fopen (“pedro”,”r”);
scanf (fich, “%d”,&edad);
fclose(fich);
/*modo lectura*/
/*lectura del fichero*/

fich = fopen (“datos”,”a”);
fprintf (fich, “Pedro tiene %d años.\n”edad);
fclose (fich);
/*modo añadir*//*escribir en fichero*/
return 0;
}
```

Lectura y Escritura de caracteres en un Archivo

Los formatos de las instrucciones de lectura y escritura de son:

carácter_leido = fgetc (fichero);

fgetc lee un carácter del fichero, el carácter leído se almacenará en carácter leído. Cuando se llega al final del fichero devuelve EOF.

fputc (car, fichero);

fputc escribe el carácter car en el fichero. Devuelve el carácter escrito o EOF en caso de error.

Lectura y Escritura de cadenas en un Archivo

Las funciones fgets() y fputs() pueden leer y escribir cadenas a o desde los archivos. Los prototipos de estas funciones son:

```
char *fputs(char *str, FILE *F);
```

La función puts() escribe la cadena a un archivo especifico.

Lectura y Escritura de cadenas en un Archivo

char *fgets(char *str, int long, FILE *F);

La función fgets() lee una cadena desde el archivo especificado hasta que lee un carácter de nueva línea o longitud-1 caracteres.

fgets (cadena_leida, num_caracteres, fichero);

Lee num_caracteres del fichero y los almacena en cadena_leida colocando el carácter de fin de cadena '\0' en la posición num_caracteres de la cadena leida.

Ejemplo: leer un carácter del archivo nombres.dat.

```
……..
char letra;
FILE *datos;datos = fopen (“nombres.dat”,”r”);

letra = getc (datos);
fclose (datos);
……..
```

Ejemplo: escritura de la letra “X” en el archivo nombres.dat.

```
……..
char letra = ‘X’;FILE *datos;
datos = fopen (“nombres.dat”,”w”);
putc (letra,datos);
fclose (datos);
……..
```

Ejemplo: Programa que copia un archivo carácter a carácter. Realizar las correcciones que copnsidere pertinente.

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

void main()
{   FILE *f1, *f2;
    char nombre[30], c;

    printf("Introduzca el nombre del fichero a copiar: ");
    gets(nombre);

    if ((f1 = fopen(nombre, "r"))==NULL)
      printf("No se puede abrir el archivo");
    else if (f2 = fopen("Copia.dat", "w")==NULL)
      printf("No se ha podido crear el fichero copia.");
    else
     {   c = getc(f1);
         while (!feof(f1))
         {   putc(c, f2);
             c = getc(f1);
         }
         fclose(f1);
         fclose(f2);
     }
  return 0;
}
```

Función feof ()

La función feof(), que determina cuando se ha alcanzado el fin del archivo leyendo datos binarios. La función tiene el siguiente prototipo:

```
int feof(FILE *F);
```

Su prototipo se encuentra en STDIO.H. Devuelve cierto si se ha alcanzado el final del archivo, en cualquier otro caso, 0. Por supuesto, se puede aplicar este método a archivos de texto también.

Función remove ()

La función remove() borra el archivo especificado. Su prototipo es el siguiente:

```
int remove(char *nombre_archivo);
```

Devuelve cero si tiene éxito. Si no un valor distinto de cero.

Función fseek ()

El **Acceso Directo** implica la posibilidad de acceder a un elemento determinado sin tener que acceder a los precedentes. Para realizarlo es utiliza la función `fseek()`, que sitúa el indicador de posición del archivo donde se le indique, siempre que sea correcta.

Este tipo de acceso lo utilizaremos **sólo en ficheros binarios**, pues existen casos en los que esta operación no funciona correctamente con ficheros de texto.

La función fseek ( ) permite desplazar el indicador de posición del archivo a la posición que se le indique.

Función fseek ()

El prototipo de la función es:

int *fseek* (FILE **arch* , long *desplazamiento* , int *origen*)

Donde:

arch	**Puntero a la estructura FILE asociada con el archivo**
desplazamiento	**es la cantidad de bytes que se desplazará el indicador de posición a partir de *origen.*Pueden ser: Positivo (movimiento adelante)Negativo (movimiento atrás)**
origen	**es una constante que determina el punto de referencia a partir del cuál se realiza el desplazamiento.**

Valor de Retorno :devuelve un valor para que el programa pueda controlar el correcto funcionamiento. Este valor puede ser:0: Todo ha funcionado correctamente-1: En otro caso.

Función fseek () – Ejemplo

Ejemplo:

Para posicionarse al comienzo del fichero:

```
fseek (fich, 0L, 0);
```

Para posicionarse al final del fichero:

```
fseek (fich, 0L, 2);
```

Para posicionarse en otro punto. Ejemplo byte 210.

```
fseek (fich, 210L, 0);
```

Utilizando una variable.long vari = 210L;

```
fseek (fich, vari, 0);
```

Se puede utilizar autoincrementando o autodecrementando:

```
fseek (fich, vari++, 0);fseek (fich, vari- -, 0);
```

Función fseek ()

Los valores que se le pueden dar a origen figuran en la siguiente tabla. Dichos valores se encuentran definidos en stdio.h

SEEK_SET	**A partir del comienzo del archivo**
SEEK_CUR	**A partir de la posición actual del archivo**
SEEK_END	**A partir de el final del archivo**

Ejemplos:

fseek (ptr , 0L , SEEK_SET)

Mueve el indicador de posición al comienzo del archivo. El origen es SEEK_SET que indica el comienzo del archivo y se desplaza 0 bytes , por lo tanto queda al principio. Es aconsejable que cuando se desee llevar el indicador de posición al comienzo del archivo se utilice ***rewind*** ya que ***fseek*** no limpia el indicador de error ni el de fin de archivo , por lo tanto cuando en determinada situación se use ***fseek*** no va a dar los resultados esperados.

Ejemplos:

fseek (ptr , 0L , SEEK_END)
Mueve el indicador de posición al final del archivo. El origen es SEEK_END que indica el final del archivo y a partir de alli se desplaza 0 bytes , por lo tanto esta al final del archivo. Si se desea en algún momento agregar datos , simplemente se debe usar esta función para enviar el indicador de posición al final del archivo.

fseek (ptr , 20L , SEEK_SET)
Mueve el indicador de posición 20 bytes a partir del comienzo del archivo.

fseek (ptr , (long) (-1)*sizeof (struct x) , SEEK_CUR)
Mueve el indicador de posición una estructura para atras a partir de la posición actual. Normalmente esta forma se utiliza cuando se estan editando datos del archivo. Al realizar una búsqueda se va leyendo cada uno de los datos del archivo por medio de fread, pero cuando encontramos el dato el indicador de posición del archivo esta en el dato siguiente al que queremos modificar , con lo cual al hacer fwrite para escribirlo se modificará otro dato. Por lo tanto antes de escribir se debe mover el indicador de posición una estructura para atrás.

Función fseek ()

Lectura en Acceso directo

1. Utilizamos `fseek()` para posicionarnos.
2. Usamos `fread()` para leer del fichero desde la posición indicada previamente.

Escritura en Acceso directo

1. Utilizamos `fseek()` para posicionarnos.
2. Usamos `fwrite()` para escribir en el fichero desde la posición indicada previamente.

La escritura en una posición del fichero no inserta sino que sobrescribe el contenido del elemento en esa posición.

Función rewind ()

En la lectura y escritura de archivos el indicador de posición se actualiza automáticamente , pero existen casos , por ejemplo en las búsquedas y modificaciones sobre archivos , en los cuales se necesita mover el indicador de posición a algún lugar en particular. Para ello se cuenta con las funciones *fseek* y *rewind*. Por otra parte se cuenta con una función que permite obtener el lugar donde se encuentra el indicador de posición del archivo , esa función es *ftell.*

Esta función permite llevar el indicador de posición al comienzo del archivo. El prototipo es el siguiente:

void *rewind* (FILE **arch*)

La función rewind ubica el indicador de posición del archivo referenciado por el puntero arch al principio y limpia los indicadores de fin de archivo y error que se encuentran el la estructura FILE.

Función ftell ()

La función *ftell* permite obtener la posición actual del indicador de posición. El prototipo es el siguiente:

long *ftell* (FILE * *arch*)

Donde *arch* es el puntero a la estructura FILE asociada al archivo.

Valor retornado: Si la operación es exitosa devuelve la cantidad de bytes que hay desde el commienzo del archivo hasta el lugar en que se encuentra el indicador de posición del archivo , en caso contrario devuelve –1L (-1 como tipo long).

Ejemplo: Obtener el tamaño de un archivo en bytes

```
#include <stdio.h>
#include <conio.h>
#define ARCH        "c:\\bin.dat"

int main ()
{
        FILE  *bin;
        long int cant;
        if ((bin=fopen(ARCH,"rb"))==NULL)
        {
                printf("No se pudo abrir el archivo");
                exit(1);
        }
        fseek (bin , 0L , SEEK_END );//Se envía el indicador de posición al final del archivo
        cant=ftell (bin);
        printf("\nEl archivo tiene %ld bytes",cant);
        fclose(bin);
        getch();
        return 0;
}
```

Terminada la descripción de todas las funciones que se utilizarán sobre archivos , podemos escribir 2 ejemplos simples de búsqueda y modificación .

Operadores Punteros

Existen dos operadores especiales de punteros: * y &.

- **El operador & (operador dirección), aplicado sobre el nombre de una variable, devuelve su dirección de memoria.**

Coloca la dirección de memoria 3500 de la variable cont en punt y la variable puntero punt está ubicada en la dirección 4000. Esta dirección es una posición interna del ordenador y no tiene que ver con el valor de cont.

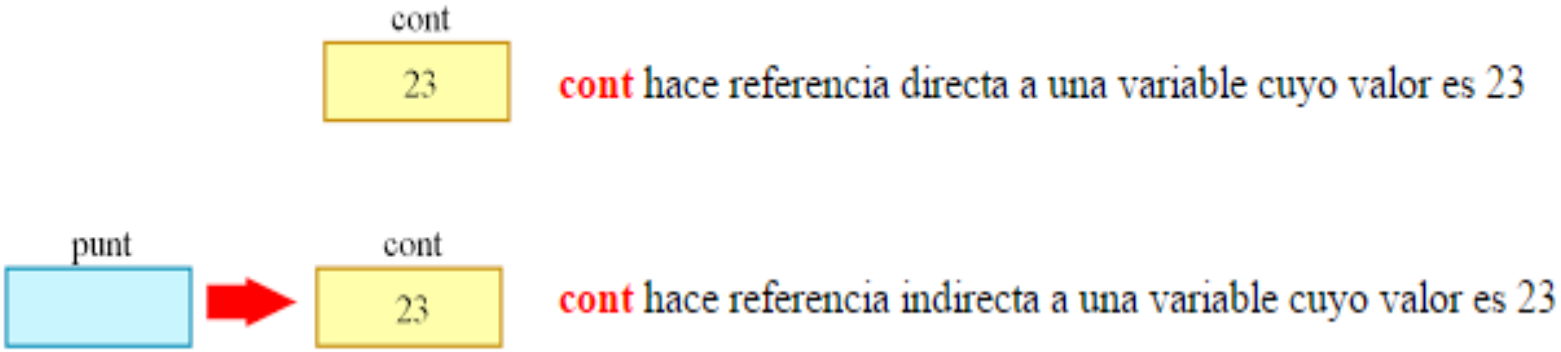
Se lee que punt "recibe la dirección del valor" cont.

Operadores Punteros

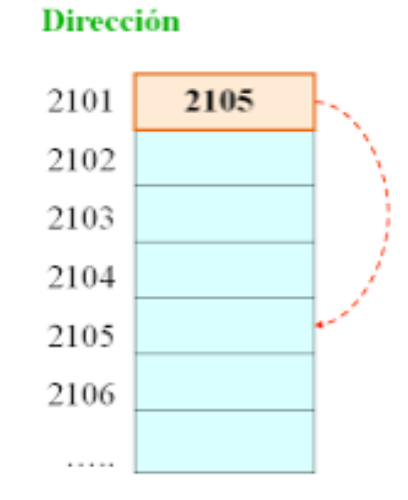
- El operador * (operador indirección) aplicado sobre una variable de tipo puntero permite acceder al dato al que apunta, es decir, al valor de la variable situada en esa dirección de memoria.

valt = *punt;

Si punt contiene la dirección de memoria de la variable cont, entonces coloca el valor de cont en valt. Ejemplo, si cont al inicio tenía el valor 23, entonces, después de esta asignación, valt tendrá el valor 23 ya que es el valor guardado en la posición 3500, que es la dirección de memoria que asignó a punt.



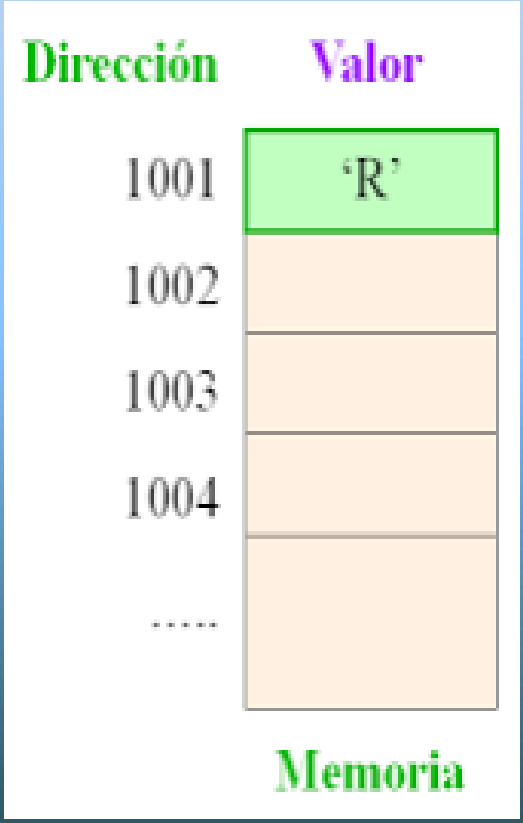
Punteros



En este sentido, los nombres de variables hacen referencia directa a un valor y los punteros hacen referencia indirecta a un valor. La referencia a un valor a través de un puntero se llama INDIRECCIÓN.

Punteros

Un puntero es una variable que tiene la dirección de memoria de otra variable que contiene un valor



La dirección es la posición que ocupa en la memoria una variable, un arreglo, una cadena, una estructura, un archivo, otro puntero, etc.

Normalmente las variables contienen valores específicos. En cambio, los punteros contienen direcciones de variables que contienen valores específicos.

Si una variable contiene la dirección de otra variable entonces se dice que: "la primera variable apunta a la segunda".

Punteros

Razones para usar punteros en la creación de programas:

- **Los punteros proporcionan los medios mediante los cuales las funciones modifican sus argumentos de llamada.**
- **Se pueden utilizar para soportar las rutinas de asignación dinámica, como pilas, colas y listas.**

Desventajas del uso de punteros:

- **Pueden provocar fallos cuando no estar bien inicializados o mal definidos.**

Formato:

Pueden ser de cualquier tipo de datos. Hay punteros que contienen la dirección de variables de cualquier tipo.

```
tipo_dato    *   Nombre_Variable;
```

Indicador de puntero