Tema 8: Registros y Archivos

En las secciones 1 a 3 se brinda una síntesis de los principales conceptos referidos al almacenamiento de datos en archivos. En la sección 4, se detallan los conceptos referidos al procesamiento de archivos en C y los conceptos referidos a la estructura de datos Registros, y en la sección 5 se presentan ejemplos de programas en C que utilizan archivos y registros.

1. Conceptos básicos

En la actualidad, para poder cumplir con sus objetivos, cualquier empresa u organización necesita almacenar y manejar grandes cantidades de datos. Por ejemplo, se necesitan los datos de los empleados, de los clientes, de los proveedores, de los productos almacenados, etc.

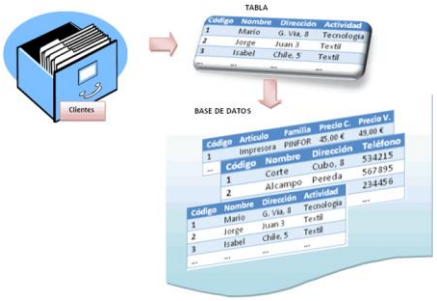
En un mundo globalizado, de alta incertidumbre y competitivo, la gestión de la información se convierte en una forma de marcar la diferencia y hacer ventaja competitiva. La utilización de las computadoras en la administración de las empresas ha cambiado el concepto de almacenamiento y gestión de sus datos, dando lugar al uso de los denominados **archivos informáticos** y **bases de datos.**

En el contexto informático, un archivo es cualquier información permanente que se almacena de cualquier forma en algún dispositivo de almacenamiento secundario, que es tratado como una unidad por el sistema operativo. Este incluye órdenes tales como *crear*, *copiar* y *borrar* archivos. Normalmente a cada fichero o archivo se le asocia un **nombre** y una **extensión** que lo identifica y diferencia del resto. Otros datos importantes asociados al archivo son: la fecha y hora de creación y actualización, el tamaño, permisos de acceso, etc. Ejemplo: La Fig. 8.1 muestra distintos tipos de archivos, y los valores de sus atributos.

Nombre	Tamaño	Tipo	Modificado
130804m.zip	2 KB	WinZip File	13/08/04 12:18 p.m.
Doce.txt	1 KB	Documento de texto	17/02/04 05:00 p.m.
Fast.ini	1 KB	Opciones de configuración	01/04/04 01:05 a.m.
Treeinfo.ncd	18 KB	Nero Cover Designer Document	06/10/04 11:27 a.m.
Viejos.prn	4 KB	Archivo PRN	17/09/04 12:21 p.m.
Whoami.msg	6 KB	Elemento de Outlook	06/10/94 04:54 p.m.

Figura 8.1. Diferentes tipos de archivos

Los archivos pueden contener distintos tipos de información: datos, instrucciones de programas, imágenes, sonido, información de control, etc. Nuestro interés en este tema son los **archivos de datos**. Los datos no se guardan en una computadora al azar, sino que se estructuran y planifican de forma adecuada, según un determinado formato. Esta tarea es realizada habitualmente por analistas y programadores.



2. Archivos de datos

Desde el punto de vista lógico, los archivos de datos se dividen en unidades lógicas llamadas **registros**, que a su vez se dividen en **campos**. Entonces, podemos decir que:
Un **archivo o fichero** es una estructura de datos que reside en memoria secundaria, consistente en un conjunto de información del mismo tipo, agrupada en unidades de acceso denominadas **registros.**

Ejemplos de archivos:

- Archivo de historias clínicas de pacientes de un hospital
- Archivo de empleados de una empresa
- Archivo de datos meteorológicos de una localidad
- Archivo de libros de una biblioteca

Un **registro lógico** o simplemente **registro** es cada uno de las componentes del archivo, que contiene el conjunto de información correspondiente a cada elemento individual, que se accede y se trata de manera unitaria. Está constituido por uno o más elementos denominados **campos.**

Ejemplos de registros:

- La información contenida en la historia clínica de un paciente de un hospital
- Los datos referentes a un empleado determinado
- Los datos meteorológicos de un día
- Los datos referidos a un libro

Un **campo** es un conjunto de caracteres que constituye un dato del objeto o entidad cuya información se almacena.

Ejemplos de campos:

- El nombre del paciente, su edad, cada uno de sus síntomas, etc.
- El nombre del empleado, fecha de alta en la empresa, puesto que ocupa, número de legajo, etc.
- Temperatura máxima, fecha, temperatura mínima, humedad, código de identificación de la estación meteorológica, etc.
- Autor, editorial, número de páginas, año de edición, etc., de un libro

En un campo se deben establecer tres características:

- *Nombre del campo:* permite rotular el mismo.
- *Tipo de campo:* indica qué tipo de dato contiene (alfabético, numérico, alfanumérico, etc.)
- *Tamaño del campo:* cantidad de caracteres que puede almacenar.

En la figura 8.2 se muestra un ejemplo de registros y campos.

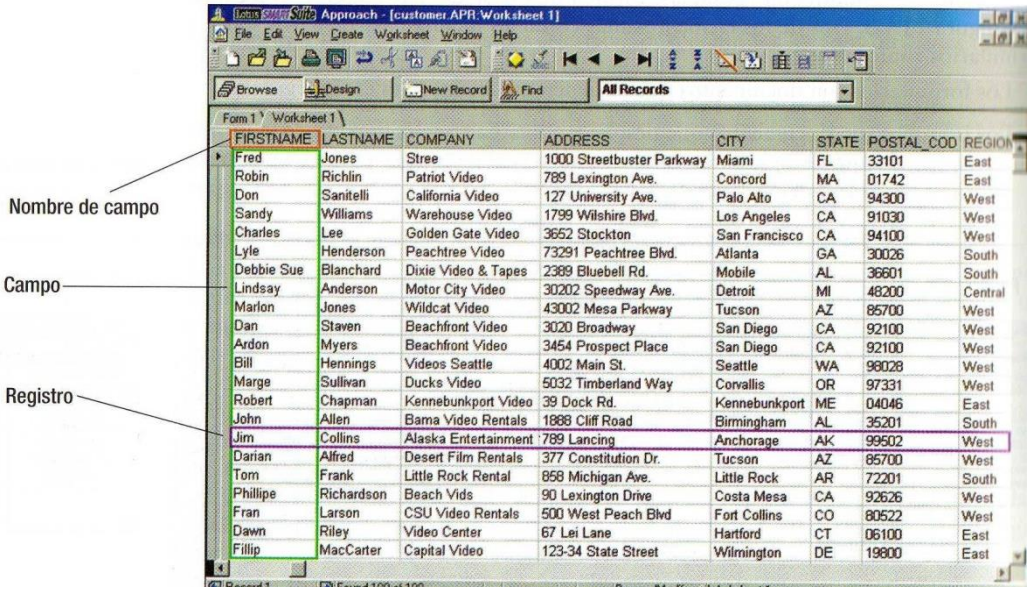


Figura 8.2. Ejemplo de registros y campos

Campo clave:

Un registro puede tener un **campo clave**, cuyo valor sirve para identificar de forma única al registro, y, por tanto, dicho valor no puede aparecer repetido en otro registro. Ejemplos: DNI cuando se trata de personas, Libreta Universitaria en el caso de los alumnos, Nº de legajo para los empleados.

Puede suceder que un archivo no tenga campo clave en sus registros o, por el contrario, que tenga varios, denominándose a la principal **clave primaria** y a las demás secundarias. Ejemplo: Un archivo puede tener como clave primaria el DNI y como clave secundaria el Nº de Libreta Universitaria. También el apellido podría ser una clave secundaria pero en este caso no habrá una identificación única, si el acceso a los datos es por Apellido se mostrarán, por ejemplo, todos los alumnos de apellido Romero.

Claves simples y claves compuestas:

Una clave *simple* está formada por el valor de un solo campo. Ejemplo: Nro. Empleado. Una clave *compuesta*, está formada por más de un campo, por ejemplo, Código de Localidad y Nro. Empleado.

Las claves se pueden utilizar para la localización rápida de los registros en archivos con determinadas organizaciones.

La figura 8.3 muestra la estructura de un archivo correspondiente a una aplicación de control de impuestos y multas de tráfico. Se puede apreciar la jerarquía de los componentes: el archivo comprende a los registros, los registros a los campos y los campos a los caracteres. Cada carácter se relaciona con el byte almacenado.

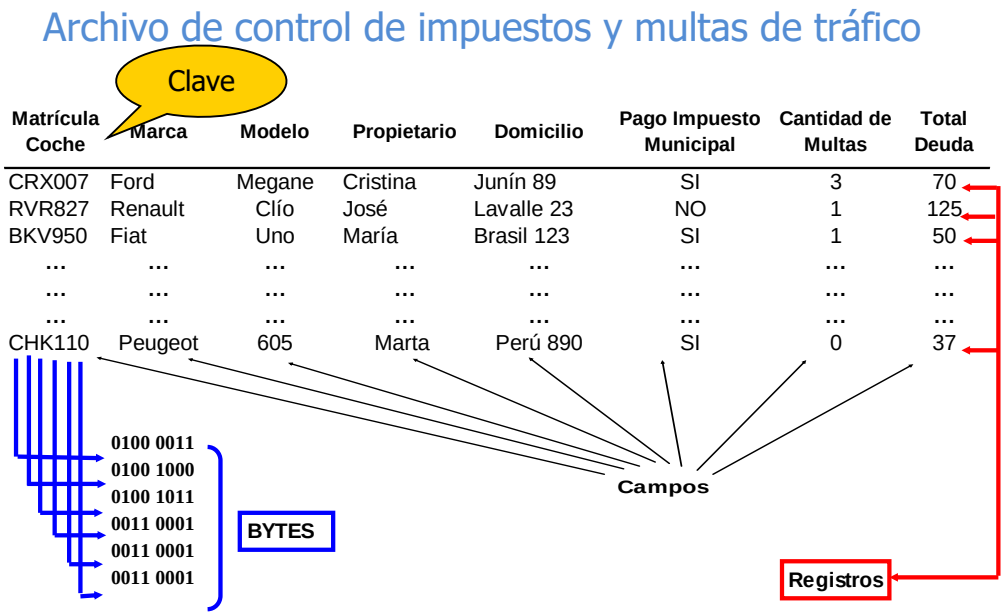


Figura 8.3. Estructura de datos de un archivo

3. Modos de acceso

Se denomina **modo de acceso** a la forma en que el dispositivo que maneja el soporte se posiciona en un determinado lugar del mismo para realizar una operación de lectura o escritura de un registro. El modo de acceso lo decide el programador de la aplicación en función del soporte utilizado y del tipo de organización.

Hay dos modos básicos: **secuencial** y **directo**.

El acceso secuencial a un registro supone acceder inicialmente al primer registro del archivo y después, consecutivamente, a todos los sucesivos hasta llegar al registro deseado. Este modo de acceso se puede utilizar con cualquier soporte y organización.

El acceso directo solamente se puede realizar en los soportes direccionables, como los discos magnéticos, y consiste en el posicionamiento sobre cualquier registro sin necesidad de haber accedido antes a los anteriores.

En los archivos de organización directa este acceso se consigue proporcionando al dispositivo la posición del registro que se desea acceder. En ocasiones es conveniente programar una función de aleatorización o *hashing*[1], la cual permite calcular la posición de cualquier registro a partir del valor de su clave.

4. Gestión de archivos

En programación, un archivo es una construcción artificial, así como el concepto de variable. La variable es un concepto abstracto que se materializa como "*contenido de una o más celdas de memoria RAM*". El concepto de archivo se materializa como "*agrupamiento organizado de bloques de memoria secundaria*".

4.1. Definición de Archivo

Un archivo, fichero o *file*, es una estructura de datos de tipo lógica, compuesta por una secuencia finita de bytes y almacenado en una memoria secundaria.

El Sistema Operativo (SO) es el encargado de interactuar con los archivos, agregando o borrando información, creando nuevos o eliminando los ya existentes. Se les debe asignar:

a) Un nombre, para su identificación.
b) Una ruta: los archivos se agrupan en estructuras y directorios (carpetas) dentro de un sistema de archivos. Los directorios poseen una estructura de árbol, con un nodo raíz y ramificaciones que representan las diferentes carpetas.

La Fig. 8.4 muestra un esquema de almacenamiento, organizado en jerarquías de carpetas que contienen a los archivos. La ruta (la jerarquía de carpetas) y el nombre identifican de forma inequívoca un archivo dentro de un sistema de archivos.

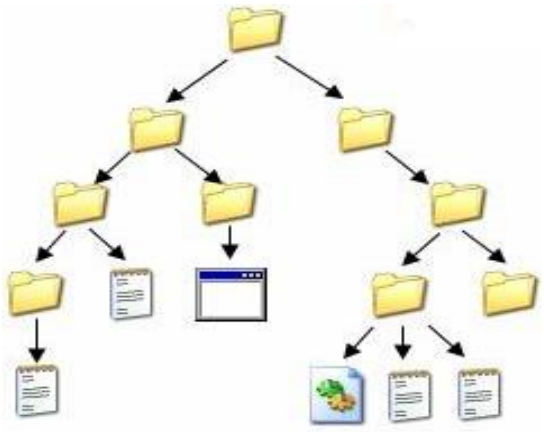


Fig. 8.4. Esquema de acceso secuencial-indexado

[1]Una función *hash* es un algoritmo de transformación de la clave del registro a una posición de almacenamiento concreta del disco, por ejemplo, si el valor de la clave es 2345 una función *hash* podría consistir en la suma de todos los dígitos de la clave, o sea, 2+3+4+5 = 14. Este resultado podría indicar la posición de almacenamiento.
.

4.2. Los archivos en C

En C hay dos tipos de archivos, **archivos de texto** y **archivos binarios**.

Un archivo de texto es una secuencia de caracteres organizados en líneas que terminan con un carácter de nueva línea **(\n).** Los archivos de texto se caracterizan por ser "planos", es decir, contienen solo texto (caracteres) sin información sobre el tipo de letra, ni formato (negrita, subrayados) ni tamaño. Técnicamente cualquier archivo puede abrirse como texto plano desde un editor de texto, por ejemplo, el Bloc de notas de Windows. Por costumbre, los archivos de texto llevan la extensión **.txt.** Se utilizan también para almacenar código fuente en programación, archivos de configuración, etc.

Un archivo binario almacena la información tal cual se representa en forma interna. En este tipo de archivos no se almacenan los números como cadenas de caracteres, sino que se almacenan de la misma forma que se hace en memoria. El contenido de los archivos binarios no puede ser visualizado mediante un editor de textos, porque los datos numéricos, por ejemplo, se representan con distintos métodos que ocupan un número determinado de bytes, por ejemplo, entero corto, entero largo, float, double.

Un fichero en C es sencillamente una colección de bytes que lleva asociado un nombre. El sistema operativo de la computadora administra los archivos por su nombre. Las operaciones que se pueden realizar sobre un archivo son:

- Abrir un archivo
- Cerrar un archivo
- Leer datos de un archivo
- Escribir datos en un archivo
- Añadir datos al final de un archivo

La diferencia entre escribir y añadir datos es la siguiente: cuando se abre un archivo para escribir datos en él, si el archivo previamente tenía datos, estos se pierden. En cambio, si se abre un archivo para agregar datos en él, los datos se agregan a continuación de los existentes, no perdiéndose ninguno.

C no tiene palabras reservadas para las operaciones de E/S. Estas se realizan a través del uso de la biblioteca de funciones. La siguiente tabla muestra las principales funciones que se utilizan para el manejo de archivos, las que están incluidas en 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
fread()	Lee un bloque de datos en archivos binarios
fwrite()	Escribe un bloque de datos en archivos binarios
feof()	Devuelve verdadero (1) si es final del archivo
ferror()	Devuelve verdadero (1) si se produce un error
fflush()	Vacía un archivo

El puntero a un archivo.

El puntero a un archivo es un puntero a información del mismo, que incluye el nombre, el estado y la posición actual del archivo. En esencia identifica un archivo específico 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 declarar una variable de este tipo se utiliza una secuencia como esta:

```
FILE *f;
```

Apertura de un archivo

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, const char modo);
```

Donde *nombre_archivo* es un puntero a una cadena de caracteres que representa un nombre válido del archivo, y puede incluir una especificación del directorio. La cadena a la que apunta *modo* determina cómo se abre el archivo. 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

La función **fopen()** devuelve un puntero a archivo. Si se produce un error cuando se está intentando abrir un archivo, **fopen()** devuelve un puntero nulo. La macro NULL está definida en **stdio.h**. Este método detecta cualquier error al abrir un archivo: como por ejemplo disco lleno, o protegido contra escritura, antes de comenzar a escribir en él. Ejemplo:

```
if ( archivoTexto == NULL ) {
      printf("El archivo No existe \n");
      exit( EXIT_FAILURE );
}
```

Si se usa **fopen()** para abrir un archivo para escritura, entonces cualquier archivo existente con el mismo nombre se borrará y se crea uno nuevo. Si no existe un archivo con el mismo nombre, entonces se creará. Si se quiere añadir al final del archivo entonces debe usar el modo “a”. Si se usa “a” y no existe el archivo, se devolverá un error.

La apertura de un archivo para las operaciones de lectura requiere que exista el archivo. Si no existe, **fopen()** devolverá un error.

Cierre de un archivo.

La función **fclose()** cierra una secuencia que fue abierta mediante una llamada a **fopen()**. Escribe toda la información que todavía se encuentra en el buffer del 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.
El prototipo de esta función es:

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

Donde F es el puntero al archivo devuelto por la llamada a fopen(). Si se devuelve un valor cero significa que la operación de cierre ha tenido éxito. Generalmente, esta función solo falla cuando un disco se ha retirado antes de tiempo o cuando no queda espacio libre en el mismo.

Leer y grabar datos en un archivo

En el procesamiento que hicimos antes, se ingresaban los datos por teclado y estos se almacenaban temporalmente en la memoria principal para su procesamiento. En tanto que, los datos almacenados en archivos residen en soportes externos a la RAM, por ejemplo, en un disco rígido, y para su procesamiento tienen que ser trasladados a la memoria RAM. Esto implica un flujo de datos entre la memoria y el dispositivo externo, tal como se muestra en la Fig. 8.5.

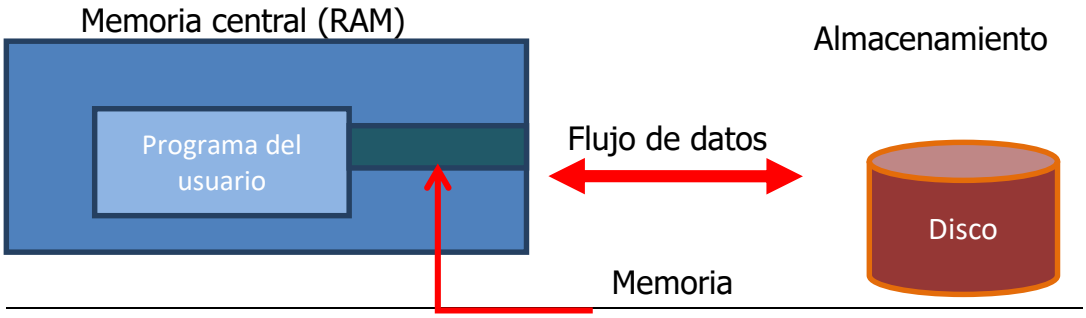


Figura 8.5: Esquema de flujo de datos entre el disco y la RAM

Para leer o grabar datos de un archivo tenemos las siguientes cuatro funciones:

fgets() y **fputs()**

Estas funciones pueden leer y escribir cadenas a o desde los archivos.
Los prototipos de estas funciones son:

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

char *fgets(char *str, intlong, FILE *F**);**

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

La función **fgets()** lee una cadena desde el archivo especificado, hasta que lee un carácter de nueva línea o longitud-1 caracteres. Si se produce un EOF (End of File) la función **gets** retorna un NULL.

fscanf() y fprintf()

Estas funciones realizan la lectura y escritura de variables escalares (es decir tipos de datos que no son compuestos) y se comportan exactamente como **scanf() y prinft()** usadas para la E/S estándar (teclado y pantalla), excepto que operan sobre archivo.
Sus prototipos son:

int fprintf (FILE *F, constchar *cadena_de_control, ...);

int fscanf (FILE *F, constchar *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.

Función feof()

La función **feof()** se utiliza para detectar cuando no quedan más elementos en un archivo, devolviendo un valor distinto de cero en este caso.
El prototipo de la función es:

int feof (FILE *F);

Devuelve verdadero (**true**) si detecta fin de archivo. En cualquier otro caso, devuelve 0. Se aplica de la misma forma a archivos de texto y binarios.

Tener en cuenta que esta función no indica fin de fichero hasta que no se intenta volver a leer después de haber leído el último elemento del archivo. Por lo tanto, en el programa debe hacerse la lectura siguiendo estos pasos:

```
Leer (obtener) un registro del archivo
mientras (!feof(archivo)
     Procesar el registro leído
     Leer (obtener) un registro del archivo
fin-mientras
```

De este modo se lee/obtiene un registro y solo se procesa si se verifica que lo que se ha leído no es la marca de fin de archivo. Esta técnica se llama de *lectura adelantada.*

ferror()

La función **ferror()** determina si se ha producido un error en una operación sobre un archivo. Su prototipo es:

int ferror(FILE *F);

donde F es un puntero a un archivo válido. Devuelve ***true*** si se ha producido un error durante la última operación sobre el archivo. En caso contrario, devuelve ***false***. Debido a que cada operación sobre el archivo actualiza la condición de error, se debe llamar a **ferror()** inmediatamente después de la operación de este tipo; si no se hace así, el error puede perderse.

fflush()

La función escribe todos los datos almacenados en el buffer sobre el archivo asociado con un apuntador. Su prototipo es:

int fflush(FILE *F);

Si se llama esta función con un puntero nulo se vacían los buffers de todos los archivos abiertos. Esta función devuelve cero si tiene éxito, en otro caso, devuelve EOF.

4.3. Tipo de dato “Registro”

Una estructura que permite almacenar diferentes tipos de datos bajo una misma variable se denomina **registro**. Un registro es un “contenedor” de datos de diferentes tipos.

Un registro en C se declara con la palabra reservada **struct**, cuyo formato es el siguiente:

```
struct  datos {
      tipo1 dato1:
      tipo2 dato2;
      /* …otros campos del registro */
} variable1, variable2;
```

La palabra reservada **struct** inicia la definición del registro que nos es más que una lista de declaraciones de variables entre llaves. A la palabra struct puede seguir, opcionalmente, un nombre. Los campos del registro se definen con un *tipox* y un nombre *datox*. Una vez definido el registro se declaran variables de este tipo.

Ejemplo: Supongamos que se desea almacenar en una estructura los datos de mediciones de temperaturas, por ejemplo: hora, minuto y valor de la medición en grados. Para ello se define la siguiente estructura:

```
struct medida {
      char hora:
      char minuto;
      float temperatura;
} regMedida
```

De esta manera se definió una variable compuesta, denominada regMedida, que contiene tres campos: la hora, los minutos y el valor de la temperatura medida. Para rellenar esta variable con datos se deben hacer asignaciones a cada uno de los campos del registro:

```
regMedida.hora=10;
regMedida.minuto=55;
regMedida.temperatura=37.5
```

Para utilizar los campos que componen el registro se utiliza el operador **“.”**, que conecta el nombre del registro con el nombre de campo:

```
nombre_registro.campo
```

En el apartado **4.3.1.** se puede ver un ejemplo de uso de registro en C

La declaración typedef

En C la declaración **typedef** permite crear **nuevos tipos de datos**. Por ejemplo:

```
typedef struct {
       char calle[25];
       int numero;
       int codigo_postal;
} t_domicilio;
```

Las instrucciones anteriores definen el tipo de dato **t_domicilio**
Esto permite, como se puede observar en la siguiente estructura, utilizar ese tipo de dato para crear 2 variables que son del mismo tipo: **domiActual** y **domiOrigen**.

```
typedef struct {
        int dni;
        int codCar;
        char codSex;
        t_domicilio domiActual;
        t_domicilio domiOrigen;
} tr_alumno;
```

4.3.1. Ejemplo de uso de registro:

```
#include<stdio.h>
void ingresarDatosAlumno();
void calcularPromedioNotas();

struct alumno {   /* Declaración de un registro para datos de un alumno */
      int dni;
      float parcial1;
      float parcial2;
      float promedio;
} regAlumno;

int main() {
      ingresarDatosAlumno();
      calcularPromedioNotas();
      return 0;
}

void ingresarDatosAlumno() {
      printf( "\nIngrese: \n" );
      printf( "\tDNI del alumno: " );
      scanf( "%d", &regAlumno.dni );
      printf( "\tNota 1er Parcial: " );
      scanf( "%f", &regAlumno.parcial1 );
      printf( "\tNota 2do Parcial: " );
      scanf( "%f", &regAlumno.parcial2 );
}

void calcularPromedioNotas() {
```

4.4. Estructuras y archivos: fread(), fwrite()

La lectura y escritura de datos (structs, matrices, ..), es algo más complicado que leer cadenas de textos. Para esto se utilizan las funciones fread() y fwrite().

La función **fread()** lee un número k de elementos, de N bytes cada uno, y los almacena en memoria a partir de la posición indicada por un puntero.

De forma inversa, **fwrite()** escribe en el archivo un número k de elementos, cada uno de N bytes, a partir de una posición de memoria determinada.

El formato genérico de ambos es:

```
fwrite (puntero, longitud, numelem, nomArchi);
```

```
fread (puntero, longitud, numelem, nomArchi);
```

Dónde:

- `puntero` es un puntero a la zona de memoria donde se va a iniciar la transferencia,
- `longitud` es el tamaño N en bytes de cada elemento a transferir,
- `numelem` es el número k de elementos a transferir,
- `nomArchi` es el puntero al archivo que se utiliza para la transferencia.

Para transferir registros de un archivo, la longitud se determina aplicando el operador **sizeof** a la estructura, que devuelve el número de bytes que la compone, y en *numelem* el valor 1 dado que se desea transferir una estructura.

4.5. Convenciones de codificación

Una buena práctica de programación es utilizar convenciones para los identificadores de los distintos componentes de un programa: constantes, variables de datos simples o estructuras de datos, tipos de datos, parámetros,

Una convención de codificación es un conjunto de directrices, normas y reglamentos sobre la forma de escribir código. Por lo general, un estándar de codificación incluye pautas sobre cómo nombrar variables, la indentación, etc.

El propósito es definir un modo constante de codificar, de modo que, si hay varias personas trabajando en el mismo proyecto de desarrollo, resulte más fácil entender lo que otros han codificado.

Incluso para el propio programador, y especialmente para los principiantes, es muy importante respetar un conjunto de reglas al escribir el código. Así, al mirar el propio código después de algún tiempo, si se ha seguido una convención de codificación, se tarda menos tiempo para comprender alguna pieza de código.

Las convenciones dan mayor legibilidad y comprensión al código y esto constituye un factor clave para facilitar el mantenimiento y la extensibilidad del software. Contribuye a desarrollar software de calidad.

Convenciones de codificación que se proponen en la asignatura:

- **Sección**
  - Declaración/Definición de constante (CONST) (**c**)
  - Declaración tipo de dato (TYPE) (**t**)
  - Declaración de variables (VAR) (**v**)
- **Tipos de dato/estructura**
  - Registro - RECORD (**r**)
  - Archivo - FILE (**f**)
- **Uso**
  - Auxiliar para almacenar temporariamente resultado de una expresión: (**a**)
  - Entrada de datos por teclado: (**e**)
  - Variable de parámetro: (**p**)

5. Ejemplos de utilización de archivos y registros en C

A continuación, se muestran ejemplos de utilización de archivos de texto y binarios (extensión .dat) y el uso de la estructura registros en C.

5.1. Archivos de texto:

A-Ejemplo de programa que graba un archivo de texto

```
#include <stdio.h>
void abrirArchivo();
void grabarArchivo();
void cerrarArchivo();

FILE * archivoTexto;

int main() {
      abrirArchivo();
      grabarArchivo();
      cerrarArchivo();
      return 0;
}

void abrirArchivo() {
      archivoTexto = fopen( "prueba.txt", "w" ); /* Se crea un archivo de
                                                    texto y se abre en modo
                                                    escritura */
      printf( "Archivo creado!\n" );
}

void grabarArchivo() {
      fputs( "Esta es una linea\n", archivoTexto );
      fputs( "Esta es otra linea", archivoTexto );
      fputs( " y esta es continuación de la anterior\n", archivoTexto );
}

void cerrarArchivo() {
```

B-Ejemplo de programa que lee un archivo de texto

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

void iniciarProceso();
void procesarArchivo();
void leerCaracter();
void mostrarCaracter();
void finalizarProceso();

FILE * archivoTexto; /* Se declara la variable "archivoTexto" como puntero
                        a un tipo FILE */
char caracter;

int main(){
      iniciarProceso();
      procesarArchivo();
      finalizarProceso();
      return 0;
}

void iniciarProceso(){
      archivoTexto = fopen( "poema20.txt", "r" ); /* Se abre el archivo
                                                     en modo lectura */

      /* Se verifica que el archivo exista */
      if ( archivoTexto == NULL ) {
            printf("El archivo No existe \n");
            exit( EXIT_FAILURE );
      }

      printf("Contenido del archivo:\n" );
      printf("----------------------\n" );
}

void procesarArchivo() {
      leerCaracter();
      while ( ! feof(archivoTexto) ) { /* itera mientras no sea fin de archivo */
            mostrarCaracter();
            leerCaracter();
      }
}

void leerCaracter(){
      caracter = fgetc( archivoTexto );      /* Se obtiene de un caracter a la
                                                vez del archivo de texto */
}

void mostrarCaracter(){
      printf( "%c", caracter ); /* Se muestra el caracter leído */
}

void finalizarProceso(){
      fclose( archivoTexto );   /* Se cierra el archivo de texto */
}
```

Nota: Al momento de la ejecución del programa el archivo “poema20.txt” debe estar en la misma carpeta en la que se encuentra el archivo ejecutable.

5.2. Archivos binarios:

A-Programa para cargar datos en un archivo de nombre ¨alumnos.dat¨.

```
#include <stdio.h>

void iniciarProceso();
void grabarArchivoAlumnos();
void finalizarProceso();

void ingresarDatosAlumno();
void grabarRegistroAlumno();
void ingresarRespuesta();

typedef struct {    /* Declaración de un tipo de dato registro */
       int dni;
       float nota1;
       float nota2;
} tr_alumno;

tr_alumno vr_alumno;
FILE * vf_alumnos; /* Declaración del archivo */
char respuesta;

int main(){
       iniciarProceso();
       grabarArchivoAlumnos();
       finalizarProceso();
       return 0;
}
void iniciarProceso() {
       /* Se abre un archivo nuevo y se asigna a la variable archivo */
       vf_alumnos = fopen("Alumnos.dat", "wb");
       printf( "\tArchivo creado de alumnos creado! \n" );
}

void grabarArchivoAlumnos() {
       ingresarRespuesta();
       while( respuesta != 'n' && respuesta != 'N' ){
              ingresarDatosAlumno();
              grabarRegistroAlumno();
              ingresarRespuesta();
       }
}

void ingresarDatosAlumno() {
       printf( "\nIngrese: \n" );
       printf( "\tDNI del alumno: " );
       scanf( "%d", &vr_alumno.dni );
       printf( "\tNota 1er Parcial: " );
       scanf( "%f", &vr_alumno.nota1 );
       printf( "\tNota 2do Parcial: " );
       scanf( "%f", &vr_alumno.nota2 );
}

void grabarRegistroAlumno() {
       /* Se graba 1 registro en el archivo de Alumnos */
       fwrite( &vr_alumno, sizeof( tr_alumno ), 1, vf_alumnos );
       printf( "\tRegistro de notas de alumno insertado! \n" );
}
void ingresarRespuesta() {
       printf( "\nDesea ingresar datos al archivo? s/n: " );
       fflush( stdin );
       scanf( "%c", &respuesta );
}
void finalizarProceso() {
       fclose( vf_alumnos ); /* Se cierra el archivo */
}
```

B- Programa para leer un archivo alumnos.dat

Lee y procesa registros de alumnos de un archivo binario.

```
#include <stdio.h>
void iniciarProceso();
void procesarAlumnos();
void finalizarProceso();

void obtenerAlumno();
void procesarAlumno();
void ingresarRespuesta();

typedef struct {     /* Declaración de un tipo de dato registro */
       int dni;
       float nota1;
       float nota2;
}tr_alumno;

tr_alumno vr_alumno;
FILE * vf_alumnos; /* Declaración del archivo */
char respuesta;
int cantAlumnosRegistrados;

int main(){
       iniciarProceso();
       procesarAlumnos();
       finalizarProceso();
       return 0;
}
void iniciarProceso(){
       /* Inicializar variables auxiliares */
       cantAlumnosRegistrados = 0;

       /* Abre el archivo y asigna a la variable archivo */
       vf_alumnos = fopen( "Alumnos.dat", "rb" );

       /* Imprimir titulos si corresponde */
       printf( "\t*** NOTAS DE ALUMNOS ***\n" );
       printf( "\nDocumento\t|\tNota 1\t|\tNota 2\n" );
       printf( "-----------------------------------------------\n" );
}

void procesarAlumnos() {
       /* Lectura adelantada: Se obtiene un registro de un alumno del archivo */
       obtenerAlumno();
       while( !feof( vf_alumnos ) ) {
              procesarAlumno();
              obtenerAlumno();
       }
}

void obtenerAlumno() {
       /* Se obtiene (lee) un registro de un alumno del archivo */
       fread(&vr_alumno, sizeof(tr_alumno), 1, vf_alumnos);
}
void procesarAlumno() {
       /* En este caso el procesar un registro de un alumno consiste en:
              1. mostrar los datos del alumno y
              2. contar el alumno */
       printf( "%d\t\t%.2f\t\t%.2f\n", vr_alumno.dni, vr_alumno.nota1, vr_alumno.nota2
);
       cantAlumnosRegistrados = cantAlumnosRegistrados + 1;
}
void finalizarProceso(){
       /* Se muestran los resultados */
       printf( "\nCantidad de registros de alumnos: %d \n", cantAlumnosRegistrados );
       fclose( vf_alumnos ); /* Se cierra el archivo */
}
```