Estructuras ... Registros en C
Agrupamiento de variables standard... Tipos de datos definidos por el programador.

Las estructuras nos permiten agrupar varios datos, aunque sean de distinto tipo, que mantengan algún tipo de relación, y manipularlos todos juntos, con un mismo identificador, o por separado.

Una estructura es un nuevo tipo de dato, definido o diseñado por el programador para cubrir la necesidad de albergar bajo un mismo identificador un conjunto de datos de diferentes tipos, pero relacionados entre sí, tales que, los miembros o elementos de la estructura están unidos de una forma lógica desde el punto de vista de la información que tratan, por lo general, a través de ellos se describe a una entidad (una persona, una empresa, etc.).

Las estructuras son llamadas también muy a menudo registros, o en inglés "records". Y son estructuras análogas en muchos aspectos a los registros de bases de datos. Y siguiendo la misma analogía, cada variable de una estructura se denomina a menudo campo, o "field"

Para poder emplear una estructura es necesario hacer una declaración de la misma mediante la palabra reservada struct, estableciendo así un patrón o plantilla de la estructura, que detalla los miembros o elementos que la componen.

Declaración:

```
struct nombre_estructura {
    tipo elemento_1;
    tipo elemento_2;
    . . . . . .. . ;
    tipo elemento_n;
};
```

```
struct empleado {
      int idEmpleado;
      char nombre[50];
      char direccion[50];
      long telefono;
      float sueldo;
};
```

Con la sentencia del ejemplo se ha realizado la declaración de una plantilla o patrón del tipo empleado, pero todavía no se ha definido en memoria ninguna variable. La declaración suele hacerse de forma global para permitir la definición de las variables del tipo estructura en los diversos módulos o funciones, sin tener que volver a declarar la plantilla en cada función.

En el interior de una estructura, entre las llaves, se pueden definir todos los elementos que consideremos necesarios, del mismo modo que se declaran las variables.

Las estructuras pueden referenciarse completas, usando su nombre, como hacemos con las variables que ya conocemos, y también se puede acceder a los elementos en el interior de la estructura usando el operador de selección (.), un punto.

Definición de variables del tipo de la estructura:

Existen dos formas para realizar la definición de las variables:

a. En la declaración de la plantilla o patrón:

```
struct empleado{
    char nombre[30];
    char direccion[50];
    long telefono;
} emple_1, emple_2, emple_3;
```

Con esta sentencia se ha declarado una estructura del tipo empleado y a la vez se han definido en memoria tres variables emple_1, emple_2 y emple_3 que contienen los elementos del tipo declarado.

Existe un caso particular en el que se puede suprimir el nombre de la estructura cuando se definen las variables en la declaración.

```
struct {
    tipo elemento_1;
    tipo elemento_2;
    ....................;
    tipo elemento_n;
} listaDeVariables;
```

```
struct {
    char nombre[30];
    char direccion[50];
    long telefono;
    float sueldo;
} emple_1, emple_2, emple3;
```

Con esta sentencia, en el ejemplo se han definido tres variables, pero más adelante no se pueden volver a definir otras variables pues no se tiene el nombre de la plantilla (estructura).

Es conveniente por tanto especificar el nombre para poder utilizarlo en cualquier función.

El nombre de la estructura es un nombre opcional para referirse a la misma. Las variables de estructura son variables declaradas del tipo de la estructura, y su inclusión también es opcional. Sin bien, al menos uno de estos elementos debe existir, aunque ambos sean opcionales.

b. Después de haber realizado la declaración de las estructuras:

```
//Formato:
struct nombre_estructura lista_de_variables ;
//Ejemplo:
struct empleado emple_4 , emple_5 , emple_6 ;
```

Con esta sentencia se han definido en memoria tres variables del tipo empleado: emple_4, emple_5 y emple_6 (estructura declarada anteriormente).

Definición de nuevos tipos de variable (typedef)

A veces resulta conveniente crear otros tipos de variables, o redefinir con otro nombre las existentes. Esto se puede realizar mediante la palabra reservada “typedef”, por ejemplo:

```
typedef unsigned long int enorme;
```

A partir de esta definición, las que siguen tienen idéntico significado:

```
unsigned long int nombre_de_variable;
enorme nombre_de_variable;
```

Uso de la sentencia “typedef” para la declaración de estructuras:

Algunos programadores prefieren el uso de “typedef” para la declaración de estructuras, ya que a veces reduce el trabajo de tipeo. Puede usarse de la siguiente manera:

```
typedef struct {
    tipo elemento_1;
    tipo elemento_2;
    ..................;
    tipo elemento_n;
} nombreDeVariable;
```

```
typedef struct {
    int idEmpleado;
    char nombre[30];
    char direccion[30];
    long telefono;
    float sueldo;
} empleado;
```

```
//Formato:
nombre_de_variable lista_de_variables;
//Ejemplo:
empleado emple_1, emple_2, emple_3;
```

Con “typedef” creamos un nuevo tipo de variable, denominada empleado, que tiene la forma de la estructura. Luego podemos definir variables de este tipo, de la manera habitual.

Más sobre las variables que usaremos del tipo de la estructura

La estructura permite asociar datos de distinto tipo y agruparlos, creando un nuevo tipo de datos, que puede ser utilizado de forma similar a los tipos de datos básicos (los definidos por el compilador).

Una variable de tipo estructura puede copiarse sobre otra variable del mismo sin necesidad de utilizar el operador de selección (el punto), por ejemplo:

```
void main() {
    empleado unEmpleado;
    empleado otroEmpleado;
    scanf("%d", &unEmpleado.idEmpleado);
    scanf("%s", unEmpleado.nombre);
    scanf("%s", unEmpleado.direccion);
    scanf("%ld", &unEmpleado.telefono);
    scanf("%f", &unEmpleado.sueldo);
    otroEmpleado= unEmpleado; …
}
```

Esto copia el contenido de todos los campos de la variable unEmpleado en la variable otroEmpleado.

Arreglos de registros (structs):

Como se desprende de la observación de los ejemplos que venimos reproduciendo: si manejamos un conjunto de variables, todas del mismo tipo (emple_1, emple_2, ..., emple_n), es lógico suponer que es factible agruparlas en un arreglo. Es posible crear arreglos de tipos de datos structs, y aplicarles todos los algoritmos aplicables a cualquier tipo de arreglo. Esto permite un uso más simple que los arreglos paralelos. Por lo tanto en la mayoría de los casos trabajaremos con arreglos del tipo de la estructura:

```
struct empleado emple[60]; ó empleado empleAdmin[10], empleMaestranza[50];
```

Acceso a los miembros de una estructura de datos:

Una vez que se tiene una estructura creada, el acceso a cada uno de los elementos que la componen (datos), como veremos, no es muy difícil. Solo bastará con anteponer al nombre del elemento que se quiere acceder, el nombre de la variable de estructura seguido del operador '.' (punto). Siguiendo con los ejemplos anteriores:

```
//Formato:
nombreVariableEstructura.nombreElemento
```

```
//Ejemplos:
emple_1.nombre ó emple[x].telefono ó empleAdmin[x].sueldo
```

Para acceder a un carácter determinado de las cadenas de caracteres que son elementos de la estructura de datos:

```
//Formato:
nombreVariableEstructura.nombreElemento [subíndice]
```

```
//Ejemplo:
emple_1.nombre[s] ó emple[x].nombre[s] ó empleAdmin[x].nombre[s]
```

Leer valores desde el teclado:

Se utilizarán las funciones de lectura Standard, en función del tipo de dato a capturar para cada elemento, según corresponda.

```
gets (emple_1.direccion); ó scanf ("%s ", emple[x].nombre); ó
scanf ("%f", &emple_2.sueldo); ó scanf ("%ld", &empleAdmin[x].telefono);
```

Escribir/imprimir valores en pantalla:

Se utilizarán las funciones de escritura Standard, en función del tipo de dato a imprimir para cada elemento, según corresponda.

```
puts (emple_3.nombre); ó printf ("%s", emple[x].direccion); ó
printf ("%ld", emple_4.telefono); ó printf ("%.2f", emple_admin[x].sueldo);
```

Carga de datos:

a. Asignación: Es posible asignar valores iniciales a cada uno de los elementos de una estructura, sin embargo, esta es una práctica de dudosa utilidad, ya que como podrá deducirse, ante el uso de arreglos de estructura, se convierte en algo demasiado engorroso. De todas maneras, si se decide inicializar una variable de tipo estructura el proceso es similar al utilizado para la inicialización de un arreglo.

```
emple_1 = {1009, " Jose Eneene ", " Sucasa 123 ", 4805049, 2000.00 } ;
```

b. Introducción de valores por teclado: En un apartado anterior ya se ha explicado con ejemplos, para este proceso se dispone de las típicas funciones de entrada: scanf (), gets(), getc (), etc.

c. Asignación de valores luego de la inicialización: Se puede hacer asignaciones directas a cualquiera de los elementos de cualquiera de las variables de estructura, se pueden reasignar valores como se haría con cualquier otra variable y se puede operar matemática y

alfabéticamente como lo haríamos con cualquier variable en tiempo de ejecución. La única salvedad es que, al igual que con cualquier otra variable de tipo alfabético los strings deben ser tratados de la forma que corresponde, utilizando las funciones correspondientes para el manejo de cadenas de caracteres: strcmp(), strcpy(), strcat(), strlen(), etc. (Todas contenidas en <string.h>).

```
strcpy (empleAdmin[x].nombre, "Nosecuanto, Juan");
srtcpy (empleMaestranza[x].direccion, "Acaalavuelta 321");
emple_4.telefono = 4805049;
emple_5.sueldo = 1890.50;
```

Ejemplo de carga con función:

```
typedef struct {
    int legajo;
    char nombre[30];
    int anio;
}alumno;

void cargarArregloCompleto(alumno muchos[], int dim) {
    for(int i=0; i< dim; i++) {
        scanf("%d", &muchos[i].legajo);
        scanf("%s", muchos[i].nombre);
        scanf("%d", &muchos[i].anio);
    }
}

void main() {
    alumno muchosAlumnos[15];
    cargarArregloCompleto(muchosAlumno); ...
}
```

Estructuras anidadas

Dentro de una plantilla de estructura se pueden anidar otras estructuras que deben estar previamente declaradas y que participarán como un miembro o elemento más de la estructura donde se anidan.

```
struct dir {
    char calle[25];
    int numero;
    int piso;
    char dpto[4];
};
```

```
struct empleado {
    char nombre[30];
    struct dir direccion;
    long telefono;
    float sueldo;
};
```

```
struct empleado emple_8;  /* declaro una sola variable del tipo de la
estructura */
```

Para referenciar elementos de la estructura anidada se emplea sucesivamente el operador de selección punto '.'.

```
strcpy(emple_8.direccion.calle, "Delbarrio Bajo");
emple_8.direccion.numero = 1313;
emple_8.direccion.piso = 5;
strcpy(emple_8.direccion.depto , " C ");
```

Métodos de ordenamiento

Como ya hemos mencionado, es fácil suponer que este nuevo tipo de dato es realmente muy útil en colecciones de datos (arreglos de variables) y cómo dichos arreglos, son susceptibles de ser ordenados, a la hora de presentar los datos, de la misma manera podemos tratar a los datos de tipo registro, ordenándolos de acuerdo a nuestra conveniencia con cualesquiera de los métodos de ordenamiento disponibles (selección, por ejemplo). Solo debemos tener en cuenta que al efectuar dicho ordenamiento, cada variable del arreglo del tipo de la estructura contiene varios elementos y que podremos ordenar a partir de un criterio determinado por un campo de la estructura, pero se deberán ordenar todos los campos al unísono, pues lo que se cambiará será el orden de los elementos del arreglo. No hace falta copiar dato a dato, la simple asignación de una variable de tipo struct contra otra de su mismo tipo, copia toda la información en ella contenida.

Búsqueda del menor elemento de un arreglo de structs

Para esto hay que determinar qué campo es quien determinará el criterio de búsqueda, por ejemplo, buscar la posición dentro del arreglo del alumno con menor número de legajo:

```
int buscarMenorAlumno(alumno A[ ], int pos, int validos) {
    int posMenor = pos;
    alumno menor = A[pos];
//menor es una variable de tipo alumno, inicializada con la primer celda
del arreglo de tipo alumno también.
    for(int i=pos+1; i<validos; i++) {
        if( A[i].legajo< menor.legajo) {
            posMenor = i;
            menor = A[i];
        }
    }
    return posMenor;
}
```

Los algoritmos restantes sobre arreglos de structs quedan como ejercitación para el alumno.