1.- Estructura

La estructura permite manejar datos de diferentes tipos. Las uniones permiten almacenar diferentes tipos de datos en las mismas posiciones de memoria. Ambas estructuras de datos son el fundamento de las bases de datos y hojas de cálculo.
Ejemplos son: a) Un archivo de fichas con información de clientes, es una estructura de elementos relacionados; b) Un listado del directorio de una PC.

Una estructura puede ser tratada como un grupo de variables que pueden ser de diferentes tipos tratadas todas juntas como una unidad.

El lenguaje C proporciona cinco formas diferentes de crear tipos de datos personalizados.

1. La *estructura* es una agrupación de variables bajo un nombre común, llamado en ocasiones tipo de datos *compuesto* (también llamado <<agregado>> o <<conglomerado>>.
2. El *campo de bits* es una variable de la estructura que permite un fácil acceso en modo binario.
3. La *unión* permite que una misma zona de memoria se defina como dos o más tipos diferentes de datos.
4. La *enumeración* es una lista de símbolos.
5. La palabra reservada **typedef** define un nuevo nombre para un tipo ya existente.

Una *estructura* es un conjunto de variables a las que se hace referencia bajo un mismo nombre, siendo una forma eficaz de mantener agrupada una información relacionada. Una *declaración de estructura* forma una plantilla que puede utilizarse para crear estructuras de objetos. Las variables que componen la estructura se llaman miembros de la estructura (los miembros de la estructura también se llaman usualmente *elementos* o *campo)*.
En general, cada miembro de la estructura está relacionado con el resto. Por ejemplo, en una lista, la información relativa al nombre y a la dirección se representa, normalmente, como una estructura. En el siguiente fragmento de programa vemos cómo declarar una estructura que contenga los campos nombre y dirección. La palabra reservada **struct** indica al compilador que se va a definir una estructura.

El formato general de declaración de una estructura es

```
struct etiqueta{
        tipo_nombre variable;
        tipo_nombre variable;
        tipo_nombre variable;
        .
        .
        .
}variable_de_estructura;
```

La *etiqueta* es **un nombre del tipo de la estructura**, no un nombre de variable, *variables_de_estructura* es una lista de variables separadas por comas. Recuérdese que, aunque la *etiqueta* y las *variables_de_estructura* son opcionales, es obligatorio especificar, al menos, una de ellas.

Sintaxis de Estructura

Ejemplo 1:	**Ejemplo 2:**
struct correo{ char nombre[30]; char calle [40]; char ciudad [20]; char estado [3]; }; */* se **utiliza ;** para terminar la instrucción en C ´ó C++*/*	struct correo{ char nombre[30]; char calle [40]; char ciudad [20]; char estado [3]; unsigned long int codigo; }; */* se pueden utilizar diferentes tipos de variables, en este caso se combina char con int */*

El nombre de la estructura también se denomina *etiqueta*.
En los ejemplos dados no existe ninguna **variable asociada con la estructura**, lo cual se puede hacer de la forma siguiente:

Ejemplo 3:	**Ejemplo 4:**
Se puede asociar una variable con la estructura mediante la instrucción siguiente: **struct correo info_correo;**	struct correo{ char nombre[30]; char calle [40]; char ciudad [20]; char estado [3]; unsigned long int codigo; }info_correo*; /* declara una variable llamada **info_correo** del tipo de estructura que la precede */*
Info_correo está declarada como estructura de tipo ***correo***. Se puede omitir la etiqueta de la estructura cuando haya una sola variable asociada con el tipo de estructura. **Ejemplo 5:** struct { char nombre[30]; char calle [40]; char ciudad [20]; char estado [3]; unsigned long int codigo; }info_correo*;*	

Referencia a miembros de una estructura

Para hacer referencia a cada miembro de una estructura se utiliza el operador (**.**) – generalmente denominado <<punto>>. Por ejemplo, la siguiente instrucción asigna el código postal 4000 al campo **código_postal** de la estructura **info_correo** declarada anteriormente.

info_correo.codigo = 4000;

El formato general es:

nombre_de_estructura.nombre_de_miembro;

Así, para escribir el código postal en la pantalla se debe poner:

printf("%d",info_correo.código);

De igual formal, la cadena de caracteres **info_correo.calle** se puede utilizar en una llamada a **gets()**, de esta manera:

gets(info_correo.calle);

por lo tanto ***info_correo*** es el nombre asociado con la estructura y **calle** es un miembro de ésta.

Asignaciones de estructuras

La información contenida en una estructura se puede asignar a otra del mismo tipo mediante una única instrucción de asignación. Es decir, no es necesario asignar valor a valor cada miembro de la estructura.

P.N° 1:
Realizar un programa que muestre cómo asignar contenidos a una estructura y como asignar una estructura a otra estructura.

```
#include <stdio.h>

int main(void){
        struct{
                int a;
                int b;
        }x, y;

x.a  = 10;
x.b  = 20;
y = x; /*asigna una estructura a otra*/

printf("Contenido de y: %d %d", y.a, y.b);
return 0;}
```

Tras la asignación, **y.a** e **y.b** contienen los valores 10 y 20, respectivamente.

1.2.- Arrays de estructuras

La utilización más común de las estructuras son los *arrays de estructuras.* Para declarar un array de estructuras, se debe definir primero la estructura y, a continuación, declarar una variable array de dicho tipo. Por ejemplo, para declarar un array de 10 elementos del tipo correo (declarado anteriormente) se debería hacer de la siguiente forma:

struct correo info_correo[10];

Para acceder a una determinada estructura del array **info_correo**, se indexa el nombre de la variable array. Por ejemplo para mostrar el código postal de la tercera estructura, se puede hacer:

printf("%ld", info_correo[2].código);

Ejemplo de inventario
La información a almacenar será:

- Nombre del artículo
- Costo
- Cantidad en existencia

Para almacenar esta información definimos una estructura:

```
#define MAX 100

struct inv{
        char articulo[30];
        float costo;
        int existencias;
}info_inv[MAX];
```

2.- Tipos de datos definidos por el usuario (typedef)

La característica ***typedef*** permite definir nuevos tipos de datos que sean equivalentes a los tipos de datos existentes.
Una vez definido el tipo de datos por el usuario, las nuevas variables, arrays, estructuras, pueden se declaradas en términos de este nuevo tipo de datos

Un nuevo tipo de datos se refiere como:

typedef tipo nuevo-tipo;

donde tipo se refiera a un tipo de datos existente (un tipo de dato estándar o previamente definido por el usuario) como pueden ser los casos de int y float, y nuevo-tipo definido por el usuario, con la particularidad que el nuevo tipo es nuevo sólo de nombre ya que no será fundamentalmente diferente de los tipos de datos estándar.

Ejemplo 6:
El programa crea dos tipos nuevos “entero” y 2 real”, que se van a usar en lugar de “int” y “float”

```
#include <stdio>
typedef int entero;
typedef float real;

void main (void)
{
entero  i= 567;
```

```
real      mireal=3.1459;

printf ( " El número entero es %d".\n", i);
printf (" El número real es %f.\n", mireal);
getc();

}
```

3.- Bibliografía Recomendada

- Byron S. Gottfried: " Programación en C". México. Serie Schaum – McGraw – Hill.
- Chris H. Pappas / William H. Murray III: "Manual de Borland C^{++} ". Versión 3.1. México. Editorial McGraw – Hill.
- Herbert Schildt. "C-Guía de Autoenseñanza". Editorial McGraw – Hill.

Mg. Ing. J.C. Colombo
13/04/14