NOMBRE DE LA MATERIA: **PROGRAMACIÓN**

NRC: **42555**

HORARIO: **MARTES Y JUEVES 9 AM – 10:55AM**

ESTUDIANTE: **EFRAIN ROBLES PULIDO**

CODIGO: **221350095**

TEMA: **REGISTROS**

FECHA: **5 DE DICIEMBRE DE 2021**

Registros

En múltiples aplicaciones informáticas es necesario establecer una estructura de datos que agrupe la información de diversos atributos de un único concepto. El tipo de datos **registro** es el que sirve entonces para guardar información de distinto tipo en una estructura única. En C los registros se representan mediante el tipo de dato denominado struct, por eso "españolizando" el término es habitual denominar estructura al tipo registro de C.

Una **estructura** es una colección de uno o más tipos de elementos denominados **miembros o campos**, cada uno de los cuales puede ser un tipo de dato diferente.

Una estructura puede contener cualquier número de miembros, cada uno de los cuales tiene un nombre único, denominado nombre de miembro.

Su sintaxis, tiene diferentes formas:

1) Se define un registro donde los diferentes campos son definidos dentro de las llaves. Este bloque solo debe contener definición de variables. Para después poner las variables que se definirán con este tipo de variable.

```
struct <nombre_estructura>
{
   <tipo> <nombre_campo1>;
   <tipo> <nombre_campo2>;
   ...
   <tipo> <nombre_campoN>;
} <variable_tipo_estructura>
```

```
struct colecciones_CD
{
   char titulo[30];
   char artista[25];
   int num_canciones;
   float precio;
   char fecha_compra[8];
} cd1, cd2, cd3;
```

2) Define los las variables <var1> a <varN> como de tipo: struct <nombre>. Se puede definir una variable de tipo, de una estructura dada, donde quiera que la estructura este definida, sin repetir todo el bloque de estructura:

```
struct <nombre_estructura>
{
   <tipo> <nombre_campo1>;
   <tipo> <nombre_campo2>;
   ...
   <tipo> <nombre_campoN>;
};
struct <nombre_estructura> <variable1> , <varible2>, ... ,
```

```
struct corredor
{
   char nombre[40];
   int edad;
   char sexo;
   char categoria[20];
   char club[26];
   float tiempo;
};
struct corredor v1, s1, c1;
```

3) Es una abreviación de 1 y 2, utilizando el *typedef*, el cual se puede usar como cualquier otro tipo de dato predefinido en C al declarar las variables alias de tipo la estructura.

```
typedef struct <nombre_estructura>
{
   <tipo> <nombre_campo1>;
   <tipo> <nombre_campo2>;
   ...
   <tipo> <nombre_campoN>;
} <nombre_tipo_estructura>;
<nombre_tipo_estructura> <variable1> , <varible2>, ...
```

```
typedef struct
{
    char nombre[50+1];
    int edad;
    char sexo[9+1];
    int peso;
    int altura;
} Individuo;
Individuo juan, rosa, menores[10];
```

Operaciones con registros

Se puede acceder a una estructura, bien para almacenar información en la estructura o bien para recuperar la información de esta. Se puede acceder a los miembros de una estructura de una de estas dos formas: **1.** utilizando el operador punto (.), **2.** utilizando el operador apuntador (->).

Entonces se pueden usar para operaciones sobre un campo en particular o el paso de una estructura como parámetro. C permite pasar estructuras a funciones, bien por valor o bien por referencia utilizando el operador &. Si la estructura es grande, el tiempo necesario para copiar un parámetro struct, en el paso por valor, puede ser prohibitivo. En tales casos, se debe considerar el método de pasar la dirección de la estructura.

Paso como parámetro de entrada:

```
int mostrarIndividuo(Individuo persona)
{
  puts(persona.nombre);
  printf("%d\n",persona.edad);
  puts(persona.sexo);
  printf("%d\n",persona.peso);
  printf("%d\n",persona.altura);
}
```

Paso como valor de retorno devuelto por una función:

```
Individuo leerDatosIndividuo( )
{ Individuo persona;
  printf("Nombre: ");
  gets(persona.nombre);
  printf("Edad: ");
  scanf("%d",&persona.edad);
  printf("Sexo: ");
  gets(persona.sexo);
  printf("Peso: ");
  scanf("%d",&persona.peso);
  printf("Altura: ");
  scanf("%d",&persona.altura);
  return persona;
}
```

Paso de parámetro de salida

```
int leerDatosIndividuo(Individuo *persona)
{
  printf("Nombre: ");
  gets(persona->nombre);
  printf("Edad: ");
  scanf("%d",&persona->edad);
  printf("Sexo: ");
  gets(persona->sexo);
  printf("Peso: ");
  scanf("%d",&persona->peso);
  printf("Altura: ");
  scanf("%d",&persona->altura);
}
```

Paso como parámetro de entrada/salida

```
int modificarIndividuo(Individuo *persona)
{
   printf("Nueva edad: ");
   scanf("%d",&persona->edad);
   printf("Nuevo peso: ");
   scanf("%d",&persona->peso);
   printf("Nueva altura: ");
   scanf("%d",&persona->altura);
}
```

Asignación a los datos de un registro:

La asignación de datos a los miembros de una variable estructura se hace mediante el operador punto. La sintaxis en C es:

```
<nombre variable estructura> . <nombre miembro> = datos;
```

Algunos ejemplos:

```
strcpy(cd1.titulo,"Granada");
cd1.precio = 3450.75;
cd1.num_canciones = 7;
```

El operador punto proporciona el camino directo al miembro correspondiente. Los datos que se almacenan en un miembro dado deben ser del mismo tipo que el tipo declarado para ese miembro.

Para el operador apuntador, **->**, sirve para acceder a los datos de la estructura a partir de un apuntador. Para utilizarlo se deberá definir primero una variable apuntadora para apuntar a la estructura. La asignación de datos a estructuras utilizando el operador apuntador (puntero) tiene el formato:

```
<puntero estructura> -> <nombre miembro> = datos;
```

Así, por ejemplo, una estructura estudiante:

```
struct estudiante
{
  char Nombre[41];
  int Num_Estudiante;
  int Anyo_de_matricula;
  float Nota;
};
struct estudiante *ptr_est;
struct estudiante mejor;
ptr_est = &mejor; /* ptr_est tiene la dirección(apunta a) mejor */
strcpy(ptr_est -> Nombre, "Pepe alomdra");
ptr_est -> Num_Estudiante = 3425;
ptr_est -> Nota = 8.5;
```

Salida de datos de un registro

Para recuperar la información de una estructura se utilizara el operador de asignación o una sentencia de salida (printf(), puts(), ...). Para acceder a los miembros se utiliza el operador punto o el operador flecha (apuntador). El formato general toma una de estas dos formas:

```
<nombre variable> = <nombre variable estructura>.<nombre miembro>;
```

o bien

```
<nombre variable> = <puntero de estructura> -> <nombre miembro>;
```

Para salida:

```
printf(" ",<nombre variable estructura>.<nombre miembro>);
```

o bien

```
printf(" ",<puntero de estructura>-> <nombre miembro>);
```

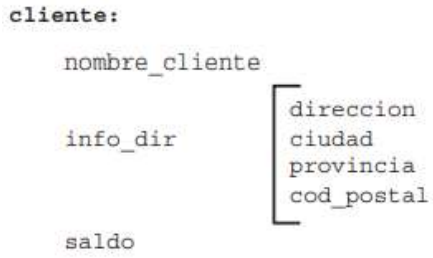
Estructuras anidadas

Nos ahorran tiempo en la escritura de programas que utilizan estructuras similares. Se han de definir los miembros comunes solo una vez en su propia estructura y a continuación utilizar esa estructura como un miembro de otra estructura. Por ejemplo:

```
struct info_dir
{
  char direccion[25];
  char ciudad[20];
  char provincia[20];
  long int cod_postal;
};
```

```
struct empleado
{
  char nombre_emp[30];
  struct info_dir direccion_emp;
  double salario;
};
```

Quedando gráficamente así:



Arreglos con elementos de tipo registros

Se puede crear un arreglo (array) de estructuras tal como se crea un arreglo de otros tipos. Los arreglos de estructuras son idóneos para almacenar un archivo completo de empleados, un archivo de inventario, o cualquier otro conjunto de datos que se adapte a un formato de estructura. La declaración de un arreglo de estructuras *info_libro* se puede hacer de un modo similar a cualquier arreglo:

struct info_libro libros[100];

Operaciones con arreglos de registros

Se podrá acceder o modificar los arreglos de registros igual como un arreglo normal, mediante un ciclo for, utilizando los modos para acceder este tipo de datos.

```
for(i=0;i<100;i++){
printf("Dame titulo del libro %d\n",i+1);
gets(libros[i].titulo);
}

for(i=0;i<100;i++)
        puts(libros[i].titulo);
```

Entrada de datos de los elementos de un arreglo de registro

Para inicializar el primer elemento de libros, por ejemplo, el código debe hacer referencia a los miembros de libros[0] de la forma siguiente:

```
strcpy(libros[0].titulo, "C++ a su alcance");
strcpy(libros[0].autor, "Luis Joyanes");
strcpy(libros[0].editorial, "McGraw-Hill");
libros[0].anyo = 1999;
```

También puede inicializarse un arreglo de estructuras en el punto de la declaración encerrando la lista de inicializadores entre llaves, { }. Por ejemplo:

```
struct info_libro libros[3] = { "C++ a su alcance", "Luis Joyanes",
"McGraw-Hill", 1999, "Estructura de datos", "Luis Joyanes",
"McGraw-Hill", 1999, "Problemas en Pascal", "Angel Hermoso",
"McGraw-Hill", 1997};
```

En el siguiente ejemplo se declara una estructura que representa a un número racional, un arreglo de números racionales es inicializado con valores al azar.

```
struct racional
{
  int N,
  int D;
};
struct racional rs[4] = { 1,2, 2,3, -4,7, 0,1};
```

Salida de datos de un arreglo de registro

Para recuperar la información de una estructura se utilizara el operador de asignación o una sentencia de salida (printf(), puts(), ...). Para acceder a los miembros se utiliza el operador punto o el operador flecha (apuntador). Pero la diferencia será que se deberá poner el índice como en un arreglo común:

```
struct nomina
{
  char nombre[30];
  int dependientes;
  char departamento[10];
  float horas_dias[7];       /* array de tipo float */
  float salario;
} empleado[100];             /* Un array de 100 empleados */
```

printf ("Nombre del empleado 3:")
puts(empleado[2].nombre)

printf ("Dependientes del empleado 3: %d", empleado[2].dependiente)

Bibliografía:

Márquez G., Osorio S., Olvera N. (2011). Introducción a la Programación Estructurada en C. Pearson

Joyanes Aguilar, L., (2014). Programación en C, C++, Java y UML. McGraw Hill. 2a. Edición

Jose, C.(2021). Registros. Consultado el 5 de diciembre de 2021 en http://lsiweb.lsi.us.es/docencia/get.php?id=6418

ISCyP, G. (2021). Arrays de Estructuras de Datos. Consultado el 5 de diciembre de 2021 en https://webs.um.es/ldaniel/iscyp17-18/17-arraysDeEstructurasDatos.html