ESTRUCTURAS

Una estructura es una colección de datos heterogéneos.

Es un componente de lenguaje C que permite agrupar datos de "distinto" tipo(concepto).

Declaración de estructuras (Sintaxis)

Como todo en lenguaje C, antes de usar algo se tiene que declarar, las estructuras se declaran de la siguiente manera:

```
struct  idEstructura
{
   <tipo1> idItem1;
   <tipo2> idItem2;
   <tipo3> idItem3;
   …
   …
   <tipoN> idItemN;
 };
```

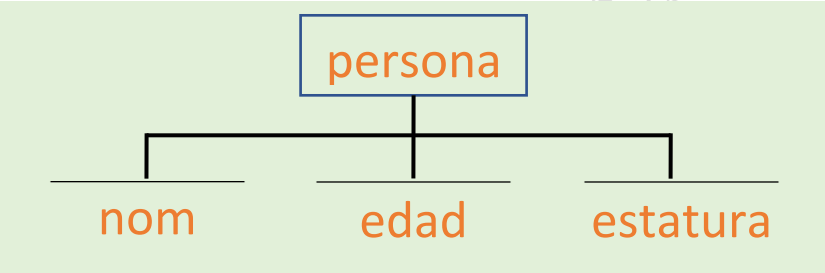
Donde los ítems (también llamados campos) son los id´s(nombres) de los componentes de la estructura y tipo1, tipo2, … son *los tipos* de cada ítem, cada tipo *puede ser* diferente.

Por ejemplo: Supongamos que queremos representar a una persona, la persona está "formada" por nombre, edad y estatura, entonces podría ser representada así:

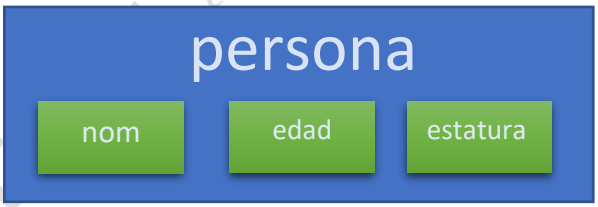
```
struct persona
{
  char nom[25];
  int edad;
  float estatura;
};
```

Nota: Esta declaración se escribiría después de la declaración de las bibliotecas (antes de los prototipos de función).

De manera gráfica la podemos representar así:



O también:



Usando la estructura

La declaración anterior equivale a "crear" un nuevo tipo de dato por lo cual no se puede usar directamente, entonces para usar la estructura se tiene que declarar una variable con ese "nuevo" tipo.

a) Declaración de *variables estructura* (Sintaxis)

```
struct idEstructura idVarEstructura;
```

Ejemplo:

```
struct persona per;
```

b) Estructuras como *parámetros*

Ejemplo:

```
___ funX(struct persona paramPer …)
```

c) Estructuras como *retorno de función*

Ejemplo:

```
struct persona funX( … )
{
    struct persona p;
    …
    return(p);
}
```

d) Estructuras como *parámetros referencia*

Ejemplo:

```
___ funX(struct persona *paramRefPer …)
```

Accediendo a los datos de la estructura

Ok, ya declaramos la variable ¿ahora qué?, ahora debemos saber ***acceder/manejar*** los datos de la estructura, como la estructura es un conjunto, debemos acceder a ellos de manera individual. El acceso se logra de la manera siguiente:

```
idVarEstructura . idItemX
```

Esta expresión nos da acceso a un elemento *particular* de la estructura. Para nuestro ejemplo sería:

per.nom per.edad y per.estatura

De esta manera accedemos al nombre, edad y estatura respectivamente, pero como ya son datos específicos podemos hacer algo como lo siguiente:

Imprimir el nombre:

```
printf("Tu nombre es: %s\n", per.nom);
```

Capturar la edad:

```
printf("Escribe tu edad: ");
scanf("%d", &per.edad);
```

Comparar la estatura:

```
if(per.estatura > 1.85)
    printf("Wow, eres muy alto");
else
    printf("Sigue creciendo");
```

Con respecto al acceso a los datos de una estructura hay que tener cuidado cuando la estructura es pasada como *parámetro por referencia*, ya que al llevar * cambia un poco la notación, veamos el ejemplo:

```
__ funX( struct per *paramRefPer ...) {
  // Dentro de la función el acceso a los elementos sería:
  (*paramRefPer) . nom     o también:   paramRefPer -> nom
}
```

En la primera expresión los paréntesis son necesarios por jerarquía, la segunda es equivalente pero un poco más simple.

Y así para todos los componentes

Bien con todos estos elementos escribamos ahora el programa completo que consistirá en un programa para manejar una ***estructura persona*** con los campos indicados anteriormente y con las *funciones básicas*: ***asignaPer***, ***capturaPer*** y ***muestraPer***.

```
/*
   Objetivo: Manejo básico de una estructura
   Autor: M.A.C.L.
   Fecha: 29/04/2021
   Nota(s): La estructura representará a una persona con los
            datos: nombre(25), edad(int), estatura(float)
 */

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

struct persona
{
  char nom[25];
  int edad;
  float estatura;
};

void asignaPer(struct persona *p, char n[], int e, float es);
void capturaPer(struct persona *p);
void muestraPer(struct persona p);

void muestraMayorEdad(struct persona p);
void cumple(struct persona *p);

int main()
{
  struct persona per;

  asignaPer(&per, "Juan Perez", 25, 1.73);
  muestraPer(per);
  capturaPer(&per);
  muestraPer(per);
  muestraMayorEdad(per);
  cumple(&per);
  muestraPer(per);
}
```

```
void asignaPer(struct persona *p, char n[], int e, float es)
{
   strcpy((*p).nom, n);
   (*p).edad = e;
   (*p).estatura = es;
}

void capturaPer(struct persona *p)
{
 printf("Dime tu nombre: ");
 gets(p->nom);
 printf("Dime tu edad: ");
 scanf("%d", &p->edad);
 printf("Dime tu estatura: ");
 scanf("%f", &p->estatura);
}

void muestraPer(struct persona p)
{
   printf("%s    %d    %.2f\n", p.nom, p.edad, p.estatura);
}

void muestraMayorEdad(struct persona p)
{
   if(p.edad > 17)
      printf("%s eres mayor de edad\n", p.nom);
   else
     printf("%s eres menor de edad\n", p.nom);
}

void cumple(struct persona *p)
{
  printf("Felicidades\n");
  p->edad++;
}
```

Como se puede ver en el programa anterior, el manejo de estructuras en un programa es una herramienta con mucho potencial, pero es un poco *laboriosa*, afortunadamente hay una forma de simplificar su manejo con el uso de la sentencia ***typedef*** que permite crear alias/apodos más cortos (o más largo según el gusto) y así simplificar la escritura.

Uso de la sentencia typedef (sintaxis)

```
typedef idActual  idNuevoAlias;
```

Ejemplo:

```
typedef struct persona PER;
```

De ahora en adelante, se puede usar PER en sustitución de struct persona.

Es posible hacer las dos declaraciones en una sola de la manera siguiente:

```
typedef struct persona
{
char nom[25];
int edad;
float estatura;
} PER;
```

Incluso se puede omitir el (id) nombre de la estructura (si no se va a usar posteriormente en el programa), declarando solo el "alias":

```
typedef struct
{
  char nom[25];
  int edad;
  float estatura;
} PER;
```

De aquí en adelante (con cualquiera de las declaraciones) podemos usar **PER** como equivalente de struct persona.

Usando esto, el programa anterior quedaría así:

```
/*
   Objetivo: Manejo básico de una estructura
   Autor: M.A.C.L.
   Fecha: 29/04/2021
   Nota(s): La estructura representará a una persona con los
            datos: nombre(25), edad(int), estatura(float)
 */

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

typedef struct
{
  char nom[25];
  int edad;
  float estatura;
} PER;

void asignaPer(PER *p, char n[], int e, float es);
void capturaPer(PER *p);
void muestraPer(PER p);

void muestraMayorEdad(PER p);
void cumple(PER *p);

int main()
{
  PER per;

  asignaPer(&per, "Juan Perez", 25, 1.73);
  muestraPer(per);
  capturaPer(&per);
  muestraPer(per);
  muestraMayorEdad(per);
  cumple(&per);
  muestraPer(per);
}
```

```
void asignaPer(PER *p, char n[], int e, float es)
{
  strcpy((*p).nom, n);
  (*p).edad = e;
  (*p).estatura = es;
}

void capturaPer(PER *p)
{
  printf("Dime tu nombre: ");
  gets(p->nom);
  printf("Dime tu edad: ");
  scanf("%d", &p->edad);
  printf("Dime tu estatura: ");
  scanf("%f", &p->estatura);
}

void muestraPer(PER p)
{
 printf("%s    %d    %.2f\n", p.nom, p.edad, p.estatura);
}

void muestraMayorEdad(PER p)
{
 if(p.edad > 17)
  printf("%s eres mayor de edad\n", p.nom);
 else
  printf("%s eres menor de edad\n", p.nom);
}

void cumple(PER *p)
{
  printf("Felicidades\n");
  p->edad++;
}
```