Repaso de Informática 1
Parte 3
Arreglos, Estructuras y Uniones

Ing José Luis MARTÍNEZ

24 de abril de 2020

1. Arreglos

En el manejo de arreglos en C, se ha estudiado que pueden ser unidimensionales y multidimensionales, por ejemplo

```
/* Ejemplo de manejo de arreglos en 1D */

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

int main(int argc, char *argv[])
{
float peso[365];// arreglo de 365 float
int cantidad[20]; // arrreglo de 20 int
char nombre[18]; // arreglo de 18 char

srand48(time(NULL));
for(int i=0; i < sizeof(peso)/4; i++)
{
float numAl = drand48()*(130.0-40.0);

    //printf("\n Numero %d: %5.2f \n", i, numAl);
    peso[i] = numAl;
   }

   for(int i=0; i<sizeof(peso)/4; i++)
    printf("\n El valor i-esimo %d de peso vale %5.2f \n", i, peso[i]);
printf("\n Tamagno del arreglo %d.\n", sizeof(peso)/4);

getchar();

return 0;
}
```

También podemos tomar matrices como arreglos

```
/*Ejemplo de arreglo en 2D - Guerra naval*/

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

int main(int argc, char *argv[])
{
char flota[10][10]={{},{}};// arreglo de 10 x 10 char
```

```
for(int i=0; i < 10; i++)
{
for(int j=0; j < 10; j++)
{
flota[i][j] = '~';
    }
   }//*/
printf("\nEscenario antes de la batalla.\n\n ") ;
for(int i=0; i < 10; i++)
{
for(int j=0; j < 10; j++)
{
printf("  %c", flota[i][j]);
    }
   printf("\n");
   }

   /* ----------------------------------------------------------
      -- Ejemplo de ubicación de un elemento con aritmética --
      -- de punteros                                          --
      ----------------------------------------------------------*/

    printf("\n\nEscenario al colocar unos barcos.\n\n ") ;
   flota[1][1]='B';
   flota[+3][+2]='B';
   flota[1+4][1+4]='B';
   *(*flota + 4*2 + 1) = 'B';

   for(int i=0; i < 10; i++)
{
for(int j=0; j < 10; j++)
{
printf("  %c", flota[i][j]);
    }
   printf("\n");
   }

getchar();

return 0;
}
```

Ejemplo Arreglo que cambia de tamaño e inserta un elemento en una posición dada. El compilador, por ejemplo gcc, suele agregar protección a las variables (llamadas canaries o canarios) que tienen valores conocidos. Una cadena de tamaño mayor a 10 puede causar corrupción de esta variable y provoca que SIGABRT (el alumno deberá investigar que es esto) finalice el programa. Cuando se está seguro de la aplicación se puede inhabilitar esta protección con una bandera en gcc usando la opción -fno-stack-protector. Pero puede llegar a aparecer otro error como una violación de segmento por tratar de acceder a una ubicación de memoria ilegal.

```
/* Escriba un programa que inserte un número en una ubicación del array.*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int i, n, num, pos, arr[10];
    printf("\n Ingrese la cantidad de elementos del array : ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
        {
            printf("\n arr[%d] = ", i);
```

```
            scanf("%d", &arr[i]);
        }
    printf("\n Ingrese el numero que desea agregar : ");
    scanf("%d", &num);
    printf("\n Ingrese la posicion en que desea agregar el numero : ");
    scanf("%d", &pos);
    for(i = n-1; i>=pos;i--)
        arr[i+1] = arr[i];
    arr[pos] = num;
    n = n+1;
    printf("\n El array despues de insertar %d es : ", num);
    for(i=0;i<n;i++)
        printf("\n arr[%d] = %d", i, arr[i]);
    getchar();
return 0;
}
```

Ejemplo Eliminar un elemento del array
Modificar el programa anterior agregando

```
 printf("\nIngrese la posición del numero a eliminar : ");
    scanf("%d", &pos);
    for(i=pos; i<n-1;i++)
        arr[i] = arr[i+1];
    n--;
    printf("\n El array después de la eliminacion es : ");
    for(i=0;i<n;i++)
        printf("\n arr[%d] = %d", i, arr[i]);
```

1.1. Práctica. Problemas a resolver con arreglos

1. Modifique el programa de guerra naval de tal forma que:
   a) Pueda agregar los barcos en forma aleatoria, según la cantidad tradicional en el juego, marcando su posición con una B.
   b) Pueda agregar los barcos en forma manual.
   c) Ingresar las coordenadas y que el juego le diga si impactó en un barco o hizo agua, en este último caso marcar la posición con una x.
   d) Modificar el escenario original que contiene solo agua, con los impactos errados (x) y los impactos en barcos (B).
2. Modifique el programa 1, para permitir la opción de multijugador, para ello cada mapa de disparos deberá guardarse en un archivo y recuperar en cada turno de jugador
3. Modifique el programa 1 y 2, para realizar un juego de guerra aero-naval. Para ello deberá agregar otra dimensión al arreglo, haciendo una matriz para barcos y otra para aviones.
4. En los sistemas electrónicos discretos y digitales, como los sistemas de control digital, procesamiento de trayectorias, ubicación espacial de un robot, trayectoria de un robot, etc; es necesario utilizar matrices y en general podemos decir que los sistemas microprocesados obtienen su mejor rendimiento con matrices. El alumno deberá implementar una serie de funciones de matrices que realicen las siguientes operaciones
   - Construir una matriz $m \times n$
   - Sumar dos matrices $C_{i,j} = A_{i,j} + B_{i,j}$
   - Restar dos matrices $C_{i,j} = A_{i,j} - B_{i,j}$
   - Obtener la trapuesta de una matriz $m \times n$, donde $A_{i,j} = B_{j,i}$
   - Obtener la matriz antisimétrica de un vector $\mathbf{x} = [a, b, c]$
   - Multiplicar dos matrices, recuerde que la cantidad de columnas de la primera matriz debe ser igual a la cantidad de filas de la segunda matriz, y que su producto no es conmutativo. Tenemos entonces que $C_{i,j} = \sum A_{i,k} B_{k,j}$ para $k = 1...n$ de una matriz $m \times n$
   - Obtener el producto escalar (<producto interno>) de dos vectores

2. Estructuras

Los arreglos unidimensionales y multidimensionales permiten agrupar gran cantidad de datos, pero tienen el inconveniente de que sólo admiten un tipo de datos por variable.

Las estructuras son variables definidas por el usuario, que permiten agrupar distintos tipos de datos en una sola variable. Podemos decir que una estructura es *una colección de datos de distintos tipos a los que se denomina miembros*. Tiene la forma:

```
struct basica
{
    tipo_de_variable_1 variable_1;
    tipo_de_variable_2 variable_2;
    int variable_3;
    float variable_4;
    char variable_5;
    ...
};
```

Ejemplo. Estructura utilizada para crear un inventario de libros

```
/* Programa que realiza un inventario de libros */
#include <stdio.h>
#include <conio.h>
#define MAXTITL 41 /* maxima longitud del titulo + 1 */
#define MAXAUTL 31 /* maxima longitud del nombre del autor + 1 */

struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
 float valor;
};



int main(int argc, char *argv[])
{
struct libro libreria; /* Se declara la variable libreria
                          como una variable de tipo libro*/

 printf("Por favor ingrese el titulo del libro. \n");
 fgets(libreria.titulo, MAXTITL, stdin); /* Se accede a la parte del titulo*/
 printf("Por favor ingrese el autor. \n");
 fgets(libreria.autor, MAXAUTL, stdin);
 printf("Por favor ingrese el valor. \n");
 scanf("%f", &libreria.valor);
 printf("%s por %s: $%.2f\n",libreria.titulo,
 libreria.autor, libreria.valor);
 printf("%s: \"%s\" ($%.2f) \n", libreria.autor,
 libreria.titulo, libreria.valor);
 printf("\n\nPresione una tecla para finalizar. \n");
 getchar();
 return 0;
}
```

A la estructura también le puedo declarar las variables que voy a utilizar con ella en el programa.

```
struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
```

```
 float valor;
}libreria;
```

También le puedo dar valores iniciales, respetando el orden en que se encuentran los datos

```
struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
 float valor;
}libreria = {"La guerra y la paz", "Leon Tolstoi", 2400};
```

O bien

```
int main(int argc, char *argv[])
{
struct libro libreria = {"La guerra y la paz", "Leon Tolstoi", 2400};
```

También puedo inicializar los miembros en cualquier orden especificando a cual de ellos me refiero

```
struct libro libreria = {.autor = "Leon Tolstoi",
 .valor = 2400,
 .titulo = "La guerra y la paz"};
```

Al ser las estructuras variables definidas por el usuario, puedo utilizarlas como lo haría con cualquier otra variable de C, en consecuencia puedo asignarle distintos nombres de variables, o declarar un puntero a variable

```
int main(int argc, char *argv[])
{
struct libro libreria,  Borges, *Kafka; // Tres variables de tipo "struct libro"
```

typedef En muchas situaciones se presenta que la estructura creada va a ser utilizada en diversas partes del programa con nombres distintos, la palabra clave typedef provee un mecanismo para crear *sinónimos* o alias para tipos de variables definidas previamente. Los nombres de variables definidas por estructuras son a menudo declaradas con *typedef* para crear instrucciones más cortas. La instrucción typedef no está solamente ligada a las estructuras sino que puede ser utilizada para cualquier tipo de datos, por ejemplo:

```
    typedef unsigned char BYTE;
BYTE x, y[10], *z;

typedef struct libro { /* plantilla de la estructura libro */
  char titulo[MAXTITL];
  char autor[MAXAUTL];
  float valor;
}Libro;

Libro libreria, Borges, *Kafka;
```

La instrucción *typedef* es similar a *#define* en cuanto permite crear una variable con su propio nombre, pero se diferencian en tres aspectos

1. A diferencia #define, typedef está limitada a dar nombres simbólicos, no puede asignar valores
2. La interpretación de typedef la realiza el compilador en vez del preprocesador
3. A pesar de las limitaciones typedef es más flexible que #define

Punteros a estructuras Podemos definir un puntero a una estructura como ya se mostró anteriormente. El acceso a los miembros de la estructura se puede realizar de dos maneras distintas.

1. ```
   Libro Kafka, *ptrKafka;
   ptrKafka = &Kafka;
   ...
   ...((*ptrKafka).titulo);
   ```
   Al tener el paréntesis una precedencia mayor que el asterisco, la instrucción funciona porque apunta primero a la dirección donde se encuentra la estructura Kafka, que es un alias de libro, y luego accede al miembro titulo.

2. La forma descrita en el item 1 si bien funciona puede llegar a ser un poco confusa de utilizar en C, sobre todo cuando tenemos estructuras anidadas. Para una mejor claridad de código C introduce el operador flecha $->$ que realiza la misma operación que el direccionamiento anterior

   ```
   Libro Kafka, *ptrKafka;
   ptrKafka = &Kafka;
   ...
   ...(ptrKafka -> titulo);
   ```

Ejemplo

```
/* Programa que realiza un inventario de libros */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define MAXTITL 41 /* maxima longitud del titulo + 1 */
#define MAXAUTL 31 /* maxima longitud del nombre del autor + 1 */

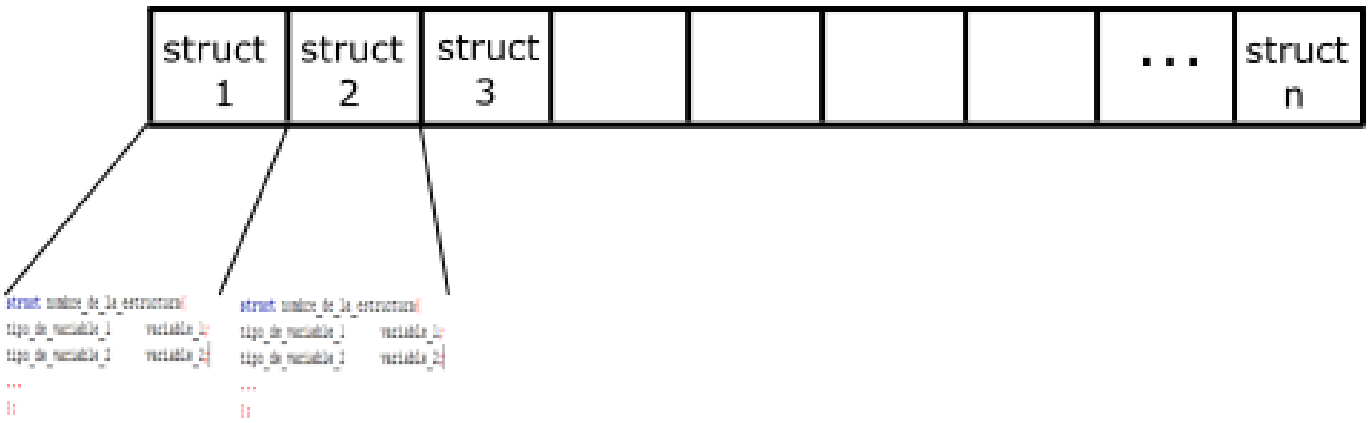
typedef struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
 float valor;
}Libro;
int main(void)
{

 Libro Kafka, *ptrKafka;
 ptrKafka = &Kafka;

  printf("Por favor ingrese el titulo del libro. \n");
 fgets(ptrKafka->titulo, MAXTITL, stdin); /* Se accede a la parte del titulo*/
 printf("Por favor ingrese el autor. \n");
 fgets(ptrKafka->autor, MAXAUTL, stdin);
 printf("Por favor ingrese el valor. \n");
 scanf("%f", &ptrKafka->valor);
 printf("\n \t Libros de Kafka ingresados. \n");
 printf("%s by %s: $%.2f\n",ptrKafka -> titulo,
 ptrKafka -> autor, ptrKafka -> valor);
 printf("%s: \"%s\" ($%.2f) \n", ptrKafka -> autor,
 ptrKafka -> titulo, ptrKafka -> valor);
 printf("\n\nPresione una tecla para finalizar. \n");


 getchar();
 return 0;
}
```

Arreglos de estructuras El ejemplo del libro sirve para agendar un solo libro lo cual hace al programa poco práctico y poco utilizable. Un bibliófilo o un melónamo necesitan tener un registro de cientos de libros o discos, pudiendo utilizar para ello un arreglo de estructuras.
Recuérdese que una estructura es un tipo de dato definido por el usuario y así como se pueden realizar arreglos de enteros de la misma forma tendremos arreglos de estructuras.
```



La declaración se hace de la misma forma que con un arreglo convencional

```
struct libro Libreria[500];
/* reserva 500 celdas capaces de almacenar la estructura, cada una de ellas */
```

Ejemplo Arreglo de estructuras para una biblioteca

```
/* Programa que realiza un inventario de libros */
#include <stdio.h>
#include <stdlib.h>
#define MAXTITL 41 /* maxima longitud del titulo + 1 */
#define MAXAUTL 31 /* maxima longitud del nombre del autor + 1 */
#define MAXLIBROS 50
typedef struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
 float valor;
}Libro;

int main(int argc, char* argv[])
{

 Libro libreria[MAXLIBROS];
 int cuenta=0;
 int indice;

 printf("Por favor ingrese el titulo del libro.\n");
 printf("Presiones [enter] al comienzo de la linea para detener la carga.\n");

 while(cuenta < MAXLIBROS && fgets(libreria[cuenta].titulo, MAXTITL, stdin) != NULL && libreria[cuenta]
 {
    printf("Por favor ingrese el autor. \n");
    fgets(libreria[cuenta].autor, MAXAUTL, stdin);
    printf("Por favor ingrese el valor. \n");
    scanf("%f", &libreria[cuenta++].valor);
    while(getchar() != '\n')
        continue;
    if (cuenta < MAXLIBROS)
        printf("\n\nPor favor ingrese el proximo libro.\n");

 }

 if(cuenta > 0){
    printf("***   Este es el listado de sus libros   ***.\n\n");
    for (indice = 0; indice < cuenta; indice++){
            printf("%s por %s: $%.2f\n",libreria[indice].titulo,
            libreria[indice].autor, libreria[indice].valor);
    }
```

```
 }
 else
    printf("No tiene libros cargados en su libreria.\n\n");

 printf("\n\nPresione una tecla para finalizar. \n");
 getchar();
 return 0;
}
```

Estructuras y funciones Las estructuras como cualquier otro dato puede ser pasado a una función mediante

Estructura a función { - Pasando miembros individuales. - Pasando la estructura completa. - Pasando la dirección de la estructura.

Pasando miembros individuales

```
/* Paso de datos de una esgtructura a una función*/

#include<stdio.h>

typedef struct
{
int x;
int y;
}Punto;
void display(int, int);
int main(int argc, char* argv[])
{
 Punto p1 = {2, 3};
 display(p1.x, p1.y);

 getchar();
 return 0;
}
void display(int a, int b)
{
 printf(" Las coordenadas del punto son: %d %d", a, b);
}
```

Pasando la estructura completa

```
/* Paso de datos de una esgtructura a una función*/

#include<stdio.h>

typedef struct
{
int x;
int y;
}Punto;
void display(Punto);
int main(int argc, char* argv[])
{
 Punto p1 = {2, 3};
 display(p1);

 getchar();
 return 0;
}
```

```
void display(Punto p)
{
 printf(" Las coordenadas del punto son: %d %d", p.x, p.y);
}
```

Pasando la estructura por referencia

```
/* Paso de datos de una esgtructura a una función*/

#include<stdio.h>

typedef struct
{
int x;
int y;
}Punto;
void display(Punto *);
int main(int argc, char* argv[])
{
 Punto p1 = {2, 3}, *ptrPunto;
 ptrPunto = &p1;
 display(ptrPunto);

 getchar();
 return 0;
}
void display(Punto *p)
{
 printf(" Las coordenadas del punto son: %d %d", p->x, p->y);
}
```

Estructuras anidadas Definimos que la estructura es un tipo de dato definido por el usuario, es decir que dentro de los miembros de la estructura puede existir también un dato definido por el usuario. Si retomamos el ejemplo de la biblioteca, vamos a agregar más datos de los libros pero siguiendo una línea jerárquica, que se vea de la siguiente forma

Libro					
Titulo	Autor	Valor	Caracteristicas		
			Categoria	Cubierta	Paginas

```
struct caract{
    char categoria[15];
    char cubierta[10];
    int paginas;

};
struct libro { /* plantilla de la estructura libro */
 char titulo[MAXTITL];
 char autor[MAXAUTL];
 float valor;
 struct caract aspecto;
};
```

Cargar entonces la categoría se realizaría de la siguiente forma

```
 fgets(libreria[cuenta].aspecto.categoria, 15, stdin);
```

2.1. Práctica. Problemas a resolver con arreglos

Responda las siguiente preguntas

5. ¿ Cuál es la ventaja de utilizar estructuras?
6. ¿ Cuál es la diferencia entre una estructura y un array?
7. ¿ Cuál es la utilidad del comando *typedef*?
8. Explique con un ejemplo como inicializar una estructura
9. ¿ Es posible crear un array de estructuras?
10. Escriba un resumen sobre estructuras anidadas.

Ejercicios de programación

12. Declare una estructura que tenga la siguiente información jerárquica
    - *a*) Carrera del estudiante
    - *b*) Legajo
    - *c*) Nombre
        1) Primer Nombre
        2) Nombres intermedios
        3) Apellidos
    - *d*) Sexo
    - *e*) Fecha de nacimiento
        1) Día
        2) Mes
        3) Años
    - *f*) Calificaciones
        1) Informática 1
        2) Informática 2
        3) Inglés
13. Modifique la información del alumno para que incluya un array con la calificaciones y los promedios finales de cada materia.
14. Convierta el programa anterior en un array de estructuras para almacenar N cantidad de alumnos
15. Declare una estructura Punto que de la ubicación de un punto respecto al origen en tres dimensiones. Cada punto representa un vector.
    - *a*) Escriba un programa que permita las operaciones de suma y resta de vectores, y la multiplicación por un escalar
    - *b*) Cree un array de 500 vectores.
    - *c*) Calcule los productos escalares y vectoriales. Utilice las funciones de matrices para ello
    - *d*) Determine si dos vectores cualquiera son ortogonales entre ellos.
16. Modifique el programa de la biblioteca para pasar los datos por referencia de cada estructura a una función y cargar los datos.
17. Diseñe un programa en lenguaje C, que cargue los datos de los alumnos para llevar un registro. El programa deberá:
    - *a*) Cargar, Legajo, Apellido, Nombre, Especialidad, Asignatura.
    - *b*) Dentro de cada asignatura deberá colocar la comisión donde cursó, docente teórico, jefe de trabajos prácticos.
    - *c*) En los docentes deberá figurar las calificaciones del parcial 1, parcial 2, promedio, y su condición que puede ser: aprobación directa, promoción práctico, regular y libre
    - *d*) Utilice estructuras anidadas para el programa y paso por referencia de cada estructura para modificar los datos

Alumno											
Legajo	Apellidos	Nombres	Asignaturas								
			Curso	Docente Teórico				J.T.P.			
				T 1	T 2	Prom	Cond	P 1	P 2	Prom	Cond

3. Uniones

Las uniones son similares a las estructuras en cuanto son variables definidas por el usuario, y que constan de variables de distinto tipo. La diferencia radica en que en el caso de las *uniones* solo puede almacenar información en un campo a la vez. La declaración de una unión es similar al de una estructura.

```
union union-nombre
{
 tipo_dato var-nombre;
 tipo_dato var-nombre;
 ..................
};
```

Por ejemplo

```
union hold {
 int digit;
 double bigfl;
 char letter;
};
...
...
union hold fit; // Variable union de tipo hold
union hold save[10]; // array de 10 variables union
union hold * pu; // puntero a una variable de tipo hold
```

Las uniones se las puede inicializar de la siguiente forma

```
union hold valA;
valA.letra = 'R';
union hold valB = valA; // inicializa una unión con los valores de otra
union hold valC = {88}; // Inicializa el primer miembro de la union
union hold valD = {.bigfl = 118.2}; // Inicializa un miembro especifico
```

Al momento de utilizarlas sucede lo siguiente

```
fit.digit = 23; // 23 es almacenado en fit; 2 bytes usados
fit.bigfl = 2.0; // 23 borrado, 2.0 almacenado; 8 bytes usados
fit.letter = 'h'; // 2.0 borrado, h almacenado; 1 byte usado
```

Ejemplo Tip de programacion.
El tamaño de una union es igual al tamaño de su miembro mas grande.

```
#include <stdio.h>
union POINT
{
int x, y;
};
int main(int argc, char *argv[])
{
int i;
 union POINT points[3];
 points[0].x = 2;
 points[0].y = 3;
 points[1].x = 4;
 points[1].y = 5;
 points[2].x = 6;
 points[2].y = 7;
 for(i=0;i<3;i++)
   printf("\n Coordinates of Point[%d] are %d and %d", i, points[i].x,
 points[i].y);
 return 0;
}
```

Uniones dentro de estructuras Generalmente las uniones pueden ser muy útiles cuando se las declara dentro de una estructura. Por ejemplo en el caso que se necesite un *string* o un *int*, dependiendo de lo que el programador especifique como se muestra en el siguiente código

```
#include <stdio.h>
struct estudiante
{
  union
  {
char nombre[20];
  int legajo;
  };
 int promedio;
};


int main(int argc, char *argv[])
{
  struct estudiante estud;
  char opcion;
  printf("\n Ingrese el nombre o el legajo de un estudiante");
  printf("\n ¿Desea agregar el nombre? (Y or N): ");
  scanf("%s", &opcion);
  while(getchar() != '\n')
   continue;
  if(opcion=='y' || opcion=='Y')
  {
 printf("\n Ingrese el nombre: ");
 fgets(estud.nombre, 20, stdin);
  }
  else
  {
 printf("\n Ingrese el legajo: ");
 scanf("%d", &estud.legajo);
  }
  printf("\n Ingrese el promedio: ");
  scanf("%d", &estud.promedio);
  if(opcion =='y' || opcion =='Y')
 printf("\n Nombre: %s ", estud.nombre);
  else
 printf("\n Legajo: %d ", estud.legajo);
 printf("\n Promedio: %d", estud.promedio);
 return 0;
}
```

3.1. Práctica. Problemas a resolver con uniones

18. Escriba un programa que contenga una estructura y una unión con exactamente los mismos miembros. Con el operador *sizeof* imprima el tamaño de cada uno de ellos.

19. En el último ejemplo de uniones, con el operador *sizeof* constate como cambia el tamaño de la estructura de acuerdo a los datos que carga en la unión anidada

20. Escriba un programa que calcule el área y el perímetro de una figura geométrica, considere círculo, triángulo y rectángulo. El cálculo se debe realizar en una función que reciba por referencia la estructura. El tipo de figura y el valor de los componentes para realizar el cálculo, deben ser parte de una unión. Tenga en cuenta que el círculo necesita un componente, el rectángulo dos y el triángulo tres.