



**UNSAM**

UNIVERSIDAD  
NACIONAL DE  
SAN MARTÍN

ESCUELA DE CIENCIA Y TECNOLOGÍA

## **TRABAJO PRÁCTICO REGULARIZADOR**

GUÍA 2.

ELECTRÓNICA DIGITAL 2

2do Cuatrimestre 2022

**Profesores:**

Ingeniero Nicolás Alvarez.

Ingeniero Miguel Angel Sagreras.

**Alumna:**

Oriana Farfán.

1. Armar un código en C para mostrar por pantalla los diferentes tamaños de los tipos: char, short, int, long, float y double.

```
#include<stdio.h>

int main() {

    char charType;
    short int short_intType;
    int intType;
    long longType;
    float floatType;
    double doubleType;

    printf("Tamaños de distintos tipos de datos.\n");
    printf("-----\n");
    printf("Un char tiene %d bytes.\n", sizeof(charType));
    printf("Un short int tiene %d bytes.\n", sizeof(short_intType));
    printf("Un int tiene %d bytes.\n", sizeof(intType));
    printf("Un long tiene %d bytes.\n", sizeof(longType));
    printf("Un float tiene %d bytes.\n", sizeof(floatType));
    printf("Un double tiene %d bytes.\n", sizeof(doubleType));

    return 0;

}
```

2. Crear un vector de enteros de 10 posiciones, recorrerlo e imprimir por pantalla cada uno de sus valores. ¿Qué tamaño ocupa en memoria?

```
#include<stdio.h>

int main()
{
    int vector[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

    for(int i=0; i<10; i++)
    {
        printf("En la posicion %d se encuentra el valor: %d.\n", i, vector[i]);
    }
    printf("Tamaño del vector:%d", sizeof(vec));

    return 0;
}
```

3. Crear una función que realice la suma de dos enteros. Utilizarla para imprimir por pantalla la suma de dos enteros predefinidos.

```
#include<stdio.h>
```

```
int suma(int A, int B){  
    int resultado;  
    resultado = A + B;  
    return resultado;  
}
```

```
int main()  
{  
    int A, B, resultado;  
  
    printf("Introducir el primer entero: ");  
    scanf("%d", &A);  
  
    printf("\nIntroducir el segundo entero: ");  
    scanf("%d", &B);  
  
    resultado = suma(A, B);  
    printf("\nResultado de la suma de los dos enteros: %d", resultado);  
  
    return 0;  
}
```

4. Indicar los valores de x e y (y sus direcciones de memoria) en cada sentencia del siguiente fragmento de código (mencionar cómo se llega a la obtención de los mismos).

```
int x = 1, y = 2;  
int *ptr;
```

```
ptr = &x;  
y = *ptr;  
*ptr = 0;
```

**Código utilizado para la obtención de resultados:**

```
#include <stdio.h>
```

```
int main() {
```

```
    int x = 1, y = 2; // se declara x e y con los valores enteros de 1 y 2
```

```
    int *ptr; // se declara el puntero de tipo int
```

```

ptr = &x; // asignacion de la dirección de memoria de x al puntero ptr

printf("El valor de x es: %d, coincide con %d. Su direccion en memoria:%p, que
coincide con %p. \n", x, *ptr, ptr, &x);

printf("El valor de y es %d y su direccion en memoria: %p.\n", y, &y);

y = *ptr; // y es el valor almacenado en la direccion de memoria a la que apunta ptr

printf("Luego de y = *ptr, el valor de y pasa a ser: %d, y la del puntero ptr sigue siendo:
%p.\n", y, ptr);

*ptr = 0; //se iguala a cero el valor almacenado en el puntero ----> x=0

printf("Luego de *ptr=0, el valor de x: %d, el valor de ptr: %d, la direccion de ptr: %p.\n",
x, *ptr, ptr);

return 0;

}

```

```

El valor de x es: 1, coincide con 1, y su direccion en memoria:0x7ffcf34aebc8, que coincide con 0x7ffcf34aebc8.
El valor de y es 2 y su direccion en memoria: 0x7ffcf34aebcc.
Luego de y = *ptr, el valor de y pasa a ser: 1, y la del puntero ptr sigue siendo: 0x7ffcf34aebc8.
Luego de *ptr=0, el valor de x: 0, el valor de ptr: 0, la direccion de ptr: 0x7ffcf34aebc8.

```

5. Codificar un procedimiento que intercambie dos enteros, por medio de la utilización de punteros. Verificarlo mediante el llamado del mismo desde un código externo con impresión del resultado (valores antes y después del intercambio).

```

void swap(int *a, int *b){
    int valorTemp;
    valorTemp = *a; // el valor temporal toma el valor de apuntado por el puntero 'a' (x).
    *a = *b; // el valor apuntado por 'a' toma el valor apuntado por 'b' (y).
    *b = valorTemp; // y finalmente, el valor apuntado por 'b' ahora es el valor temporal (x).
    return;
}

main(){
    int x = 1, y = 2;
    printf("Valores originales:\tx = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("Valores nuevos:\tx = %d, y = %d\n", x, y);
}

```

```
Valores originales:      x = 1, y = 2
Valores nuevos:         x = 2, y = 1

Program finished with exit code 0
```

6. Se tiene la siguiente declaración:

```
int x[5];
```

```
int *ptr;
```

a) ¿Cómo haría para que el puntero ptr apunte a la primera posición del vector x?

```
ptr = &x[0]; // el puntero apunta al primer valor del vector x.
```

b) ¿Recorrer el vector completo utilizando incrementos en el puntero ptr?

```
for (int i=0; i<5; i++)
```

```
{
```

```
    ptr = &x[i]; // puntero apunta hacia la posición i del vector
```

```
    printf("El valor del vector apuntado por el vector es %d\n", *ptr);
```

```
    printf("La dirección de memoria apuntado por el puntero es %p\n", ptr);
```

```
}
```

c) Realizar un printf del puntero ptr para cada incremento del punto b. Indicar por qué entre valor y valor existe un salto en la secuencia.

Hecho en el punto anterior. El salto ocurre porque el puntero fue declarado como int, ocupa 4 bytes de memoria.

```
Valor apuntado por el puntero: 0
Valor del vector apuntado por el vector: 0
Direccion de memoria apuntado por el puntero: 0x7ffc6ae72690
Valor del vector apuntado por el vector: 1
Direccion de memoria apuntado por el puntero: 0x7ffc6ae72694
Valor del vector apuntado por el vector: 2
Direccion de memoria apuntado por el puntero: 0x7ffc6ae72698
Valor del vector apuntado por el vector: 3
Direccion de memoria apuntado por el puntero: 0x7ffc6ae7269c
Valor del vector apuntado por el vector: 4
Direccion de memoria apuntado por el puntero: 0x7ffc6ae726a0
```

d) Si en lugar de tener un vector de enteros (int) en el ejemplo utilizáramos un vector de chars, el salto entre valores consecutivos del punto c cambiaría? Corroborarlo.

```
#include<stdio.h>
```

```

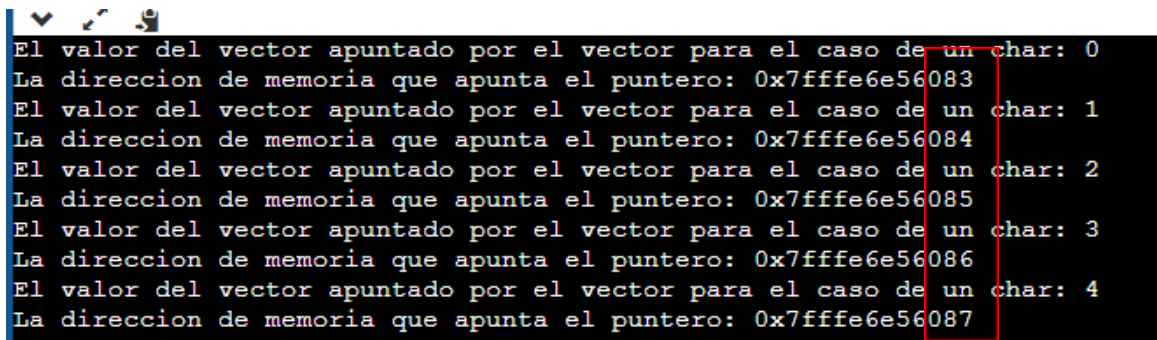
int main(){
    char c[5]={0, 1, 2, 3, 4};
    char *ptr_c;

    for (int i=0; i<5; i++)
    {
        ptr_c = &c[i]; //El puntero apunta a la direccion i del vector
        printf("El valor del vector apuntado por el vector para el caso de un char:
%d\n", *ptr_c);
        printf("La direccion de memoria que apunta el puntero: %p\n", ptr_c); //
El salto ocurre porque el puntero fue declarado como char, ocupa 1 bytes de memoria.

    }

    return 0;
}

```



```

El valor del vector apuntado por el vector para el caso de un char: 0
La direccion de memoria que apunta el puntero: 0x7fffe6e56083
El valor del vector apuntado por el vector para el caso de un char: 1
La direccion de memoria que apunta el puntero: 0x7fffe6e56084
El valor del vector apuntado por el vector para el caso de un char: 2
La direccion de memoria que apunta el puntero: 0x7fffe6e56085
El valor del vector apuntado por el vector para el caso de un char: 3
La direccion de memoria que apunta el puntero: 0x7fffe6e56086
El valor del vector apuntado por el vector para el caso de un char: 4
La direccion de memoria que apunta el puntero: 0x7fffe6e56087

```

7. Compilar el siguiente código y explicar lo obtenido en la corrida.

```

int main(void){
    int a = 0x12345678;
    short int b = 0xABCD;
    char c = 'a';
    int *ptr_a = &a;
    short int *ptr_b = &b;
    char *ptr_c = &c;
    printf("\nValor de ptr_a:\t\t%p\n", ptr_a);
    printf("Valor de ptr_a + 1:\t%p\n", ++ptr_a);
    printf("\nValor de ptr_b:\t\t%p\n", ptr_b);
    printf("Valor de ptr_b + 1:\t%p\n", ++ptr_b);
    printf("\nValor de ptr_c:\t\t%p\n", ptr_c);
    printf("Valor de ptr_c + 1:\t%p\n", ++ptr_c);
}

```

```

#include <stdio.h>

```

```

int main(void){

```

```

//Se hace la declaracion de variables
int a = 0x12345678; // se declara al entero 'a' como un numero hexa.
short int b = 0xABCD; // se declara al entero 'b' como un short hexa.
char c = 'a'; // se declara a la variable 'c' como char.

//Asignacion de punteros
int * ptr_a = &a; // puntero ptr_a apunta al valor de 'a'.
short int * ptr_b = &b; //puntero ptr_b apunta al valor de 'b'.
char * ptr_c = &c; // puntero ptr_c apunta al valor de 'c'.

//Direccion de memoria --> a
printf("\nValor de ptr_a:\t\t %p\n", ptr_a);

//Direccion de memoria siguiente a "a".
// "a" es un int que ocupa 4 bytes, se hace q la siguiente direccion disponible se
encuentre a 4 bytes de ptr_a
printf("Valor de ptr_a + 1:\t %p\n", ++ptr_a);

//Direccion de memoria --> b
printf("\nValor de ptr_b:\t\t %p\n", ptr_b);

//Direccion de memoria siguiente a "b".
// "b" es un short ocupa 2 bytes, ++ptr_b se encuentra a 2 posiciones de ptr_b.
printf("Valor de ptr_b + 1:\t %p\n", ++ptr_b);

//Direccion de memoria --> c
printf("\nValor de ptr_c:\t\t %p\n", ptr_c);

//Direccion de memoria siguiente a "c".
// Como "c" es un char ocupa 1 byte, ++ptr_c es consecutivo a ptr_c.
printf("Valor de ptr_c + 1:\t %p\n", ++ptr_c);

return 0;
}

```

8. La siguiente función calcula el largo de una cadena de caracteres. Reformularla para que la funcionalidad sea resuelta por medio del uso de un puntero auxiliar en lugar de la variable n.

```

int strlen(char *s){
    int n;

    for (n=0; *s != '\0';
        s++) n++;
    return n;
}

```

```

#include<stdio.h>
#define MAX 100

int strlen(char *s){
    int n;
    for (n=0; *s != '\0'; s++)
    {
        n++;
    }
    return n;
}

int lenModif(char *s){
    int n;
    char *ptr;      // se crea un puntero char.
    ptr = s; // puntero apunta a la palabra.
    while(*ptr != '\0')      // cuenta hasta que llegue al final.
    {
        ptr++;              // avanza a traves de la palabra.
    }
    n = ptr-s;
    return n;      // se devuelve la cantidad de veces que sumo el contador.
}

int main(){
    char palabra[MAX];    // palabra de maximo 100 caracteres.
    int num, tam;

    printf("Inserte una palabra: ");
    scanf("%s", palabra);
    num = strlen(palabra);
    printf("El largo de la cadena es: %d\n", num);

    tam = lenModif(palabra);
    printf("El largo de la cadena (con la nueva funcion): %d", tam);
    return 0;
}

```

## 9. Puntero a una estructura

Se tiene el siguiente extracto de código:



<pre> struct pru_struct {     char id1; char     id2; char     id3[10]; char     *nombre; char     *domicilio; int     edad;     int varios; };  main(){     int i;     int tmp; </pre>	<pre> struct pru_struct empleados = {     'B',     'C',     "Sensible",     "Pedro",     "Av. Carlos Calvo 1234",     23,     68, };  showinfo(&amp;empleados); </pre>
---	--

Luego de compilado se obtiene la siguiente salida:

<i>Valores iniciales de la estructura</i>	
<i>id1:</i>	<i>B</i>
<i>id2:</i>	<i>C</i>
<i>id3:</i>	<i>Sensible</i>
<i>Nombre:</i>	<i>Pedro</i>
<i>Direccion:</i>	<i>Av. Carlos Calvo 1234</i>
<i>Edad:</i>	<i>23</i>
<i>Varios:</i>	<i>68</i>
 <i>Direccion de la estructura: 0x0022FEF4</i>	
<i>Direccion del miembro id1:</i>	<i>0x0022FEF4 (offset: 0 bytes)</i>
<i>Direccion del miembro id2:</i>	<i>0x0022FEF5 (offset: 1 bytes)</i>
<i>Direccion del miembro id3:</i>	<i>0x0022FEF6 (offset: 2 bytes)</i>
<i>Direccion del miembro nombre:</i>	<i>0x0022FF00 (offset: 12 bytes)</i>
<i>Direccion del miembro domicilio:</i>	<i>0x0022FF04 (offset: 16 bytes)</i>
<i>Direccion del miembro edad:</i>	<i>0x0022FF08 (offset: 20 bytes)</i>
<i>Direccion del miembro varios:</i>	<i>0x0022FF0C (offset: 24 bytes)</i>
 <i>Dirección de la primera posición de memoria después de la estructura: 0x0022FF10</i>	

- a) Analizar los distintos valores presentes en la misma indicando claramente el significado de cada uno.

En valores iniciales de la estructura están los valores asignados a los elementos del struct.

'id3' es un array de 10 espacios, 'nombre' y 'domicilio' son punteros a una fila de caracteres.

Entre la dirección de 'id1', 'id2', e 'id3' hay saltos de memoria de 1 byte. Esto es porque esos 3 datos son del tipo char por lo que ocupan 1 byte de memoria.

Entre 'id3' y 'nombre' hay una diferencia de 10 bytes dado que 'id3' es un array de 10 es lugares, de 1 byte cada uno.

Entre la dirección de 'nombre' y 'domicilio' hay 4 espacios dado que 'domicilio' ocupa 4 bytes.

Entre 'edad' y 'varios' hay 4 espacios dado que 'edad' ocupa 4 bytes por ser un int.

- b) Completar la codificación presentada (implementar el procedimiento *showinfo*) y realizar una corrida del programa compilado, verificando los resultados obtenidos con los presentados en este punto.

```
void showinfo(struct pru_struct* empleado){
printf("\nValores iniciales de la estructura\n");
printf("\tid1:\t\t %c\n", empleado->id1 );
printf("\tid2:\t\t %c\n", empleado->id2 );
printf("\tid3:\t\t %s\n", empleado->id3 );
printf("\tNombre:\t\t %s\n", empleado->nombre );
printf("\tDireccion:\t %s\n", empleado->domicilio);
printf("\tEdad:\t\t %d\n", empleado->edad );
printf("\tVarios:\t\t %d\n", empleado->varios );

printf("\nDireccion de la estructura: 0x%p\n", empleado);
printf("\nDireccion del miembro id1:\t 0x%p (offset: %d bytes)\n", &empleado->id1, (int)offsetof(struct pru_struct, id1));
printf("\nDireccion del miembro id2:\t 0x%p (offset: %d bytes)\n", &empleado->id2, (int)offsetof(struct pru_struct, id2));
printf("\nDireccion del miembro id3:\t 0x%p (offset: %d bytes)\n", &empleado->id3, (int)offsetof(struct pru_struct, id3));
printf("\nDireccion del miembro nombre:\t 0x%p (offset: %d bytes)\n", &empleado->nombre, (int)offsetof(struct pru_struct, nombre));
printf("\nDireccion del miembro domicilio: 0x%p (offset: %d bytes)\n", &empleado->domicilio, (int)offsetof(struct pru_struct, domicilio));
printf("\nDireccion del miembro edad:\t 0x%p (offset: %d bytes)\n", &empleado->edad, (int)offsetof(struct pru_struct, edad));
printf("\nDireccion del miembro varios:\t 0x%p (offset: %d bytes)\n", &empleado->varios, (int)offsetof(struct pru_struct, varios));
printf("\nDireccion de la primera posicion de memoria despues de la estructura: 0x%p\n", ++empleado);
}
```

10. Compilar y ejecutar el siguiente código

```
#include <stdio.h>

int main(void){
char a;
int b = 0x12345678;
short int c;

printf("\n\nDireccion asignada para la variable a:\t %p\n", &a);
printf("\nDireccion asignada para la variable b:\t %p\n", &b);
printf("\nDireccion asignada para la variable c:\t %p\n", &c);
}
```

- a) Analizar los valores de salida y marcar en el gráfico siguiente (esquema de la memoria) las ubicaciones asignadas a cada variable (las posiciones de memoria son descendientes de arriba hacia abajo).

Código compilado y ejecutado:

```
Dirección asignada para la variable a: 0061FF1F
Dirección asignada para la variable b: 0061FF18
Dirección asignada para la variable c: 0061FF16
```

Variable 'a' ocupa 1 byte. Variable 'b' ocupa 4 bytes. Variable 'c' ocupa 2 bytes.

1F	a			1C
1B	b	b	b	18
17	c	c		14

- b) Analizar la asignación de la memoria en el caso en que la declaración de las variables hubiera sido hecha en el siguiente orden:

```
char a;
short int c;
int b = 0x12345678;
```

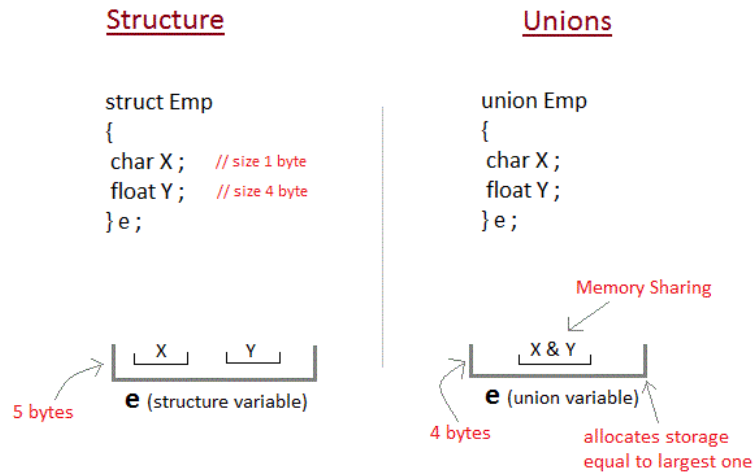
¿Se logra alguna mejora en la utilización de la memoria?

Si, al cambiar el orden de la declaración de variables se aprovechan los bits no utilizados en la declaración de 'char a' por 'short int c'. Tal que:

1F	a		c	1C
1B	b	b	b	18
17				14

# 11. Explicar la diferencia entre un **union** y un **struct**.

La diferencia está en que en la estructura, los miembros ocupan diferentes áreas de la memoria pero en la unión, los miembros ocupan la misma área de memoria.



Entonces se puede ver que: el tamaño final de la estructura va a ser la suma de cada uno de los tamaños de los miembros que la conforman, mientras que el tamaño final de la union es el tamaño del miembro de mayor tamaño que la conforme.

12. Dado el siguiente tipo de datos y teniendo en cuenta que la dirección del miembro a es 0x00546334, cuál sería la dirección de b?

```

union aux {
    int a;
    char b;
}
  
```

La dirección de b es 0x005446334.

- 13.Cuál es el tamaño de la unión del punto anterior?

El tamaño de la union es de 4 bytes.

14. Dado el siguiente código, ¿qué se mostraría por pantalla al correrlo?

```

#include <stdio.h>
union aux {
    int a;
    char b;
};

int main(void){
    union aux var;
    var.a = 77;
    printf("a: %d\n", var.a);
    printf("b: %c\n", var.b);
}
  
```

Lo que se muestra por pantalla:

```
a: 77  
b: M
```