GUÍA PRÁCTICA
Lenguaje C

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

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

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

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;
```

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){
   …
}

main(){
   int x = 1, y = 2;

   printf("Valores originales:\tx = %d, y = %d\n", x, y);
   swap(&x, &y);
   printf("Valores nuevos:\t\tx = %d, y = %d\n", x, y);
}
```

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

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.

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);
}
```

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;
}
```

9. Puntero a una estructura
   Se tiene el siguiente extracto de código:

```
struct pru_struct {
    char id1;
    char id2;
    char id3[10];
    char *nombre;
    char *domicilio;
    int edad;
    int varios;
};

main(){
    int i;
    int tmp;
```

```
    struct pru_struct empleados = {
        'B',
        'C',
        "Sensible",
        "Pedro",
        "Av. Carlos Calvo 1234",
        23,
        68,
    };

    showinfo(&empleados);

};
```

Luego de compilado se obtiene la siguiente salida:

```
Valores iniciales de la estructura
    id1:        B
    id2:        C
    id3:        Sensible
    Nombre:     Pedro
    Direccion:  Av. Carlos Calvo 1234
    Edad:       23
    Varios:     68

Direccion de la estructura: 0x0022FEF4

Direccion del miembro id1:          0x0022FEF4 (offset: 0 bytes)
Direccion del miembro id2:          0x0022FEF5 (offset: 1 bytes)
Direccion del miembro id3:          0x0022FEF6 (offset: 2 bytes)
Direccion del miembro nombre:       0x0022FF00 (offset: 12 bytes)
Direccion del miembro domicilio:    0x0022FF04 (offset: 16 bytes)
Direccion del miembro edad:         0x0022FF08 (offset: 20 bytes)
Direccion del miembro varios:       0x0022FF0C (offset: 24 bytes)

Dirección de la primera posición de memoria después de la estructura: 0x0022FF10
```

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

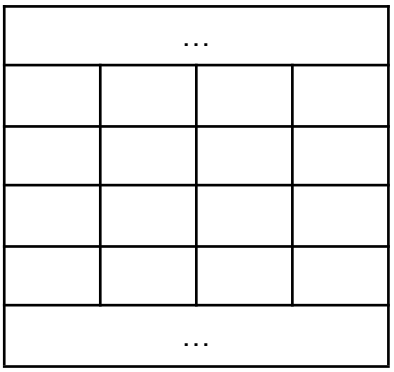
10. Compilar y ejecutar el siguiente código

```
#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).



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?

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

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;
}
```

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

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);
}
```