Laboratorio 2

Programación de Computadores (2023-2)

Temas del Laboratorio 2

- Operadores en Lenguaje C
- Estructuras condicionales
- Ciclos
- Arreglos
- Ejercicios

Operadores aritméticos en C

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo (resto de la división)

Operador Módulo %

Este operador devuelve el resto de la división entre dos números enteros:

`resto = dividendo % divisor;`

```c
int a = 5;
int b = 2;
int resto;

resto = a % b;
printf("El resto de la division de %d entre %d es %d\n", a, b, resto);
```

Output:

```
El resto de la division de 5 entre 2 es 1
```

Operadores de comparación en C

Operador	Significado
==	Igual
!=	Distinto
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

Operadores lógicos en C

Operador	Significado
&&	And (Y)
||	Or (O)
!	Not (No)

Operadores de Incremento y Decremento

Los operadores de incremento (++) y decremento (--) son operadores unarios que se utilizan para aumentar o disminuir el valor de una variable en una unidad, respectivamente.

(post-incremento) (pre-incremento)

```
variable++;  ⇔  ++variable;  ⇔  variable = variable + 1;
```

```
variable--;  ⇔  --variable;  ⇔  variable = variable - 1;
```

Diferencias entre post y pre incremento

```c
int num = 5;

printf("Valor original: %d\n", num);

printf("Resultado postincremento: %d\n", num++);
printf("Valor después de postincremento: %d\n", num);

num = 5;  // Reiniciamos el valor

printf("\nResultado preincremento: %d\n", ++num);
printf("Valor después de preincremento: %d\n", num);
```

```
Valor original: 5
Resultado postincremento: 5
Valor después de postincremento: 6

Resultado preincremento: 6
Valor después de preincremento: 6
```

Resultados de las operaciones, lvalue y rvalue

Las operaciones generan un valor que, si no se asigna a una variable, no se conserva; a estos se les llama rvalues.

Las variables a las se les puede asignar un valor son conocidas como lvalues.

L-values y R-values

```c
int num1 = 0, num2 = 0, num3 = 0;

(num1 +2) = 123;
num2++ = 123;
num3++ ++;
```

```
l_r_values.c: In function ‘main’:
l_r_values.c:7:12: error: lvalue required as left operand of assignment
    7 |  (num1 +2) = 123;
      |            ^
l_r_values.c:8:9: error: lvalue required as left operand of assignment
    8 |  num2++ = 123;
      |         ^
l_r_values.c:9:9: error: lvalue required as increment operand
    9 |  num3++ ++;
      |         ^~
```

```c
int n1 = 1;
int n2 = 1;
int a, b;

a = n1++;
b = ++n2;

printf("   a: %d    b: %d\n",a ,b);
printf("  n1: %d   n2: %d\n", n1, n2);
printf("n1++: %d ++n2: %d\n", n1++,++n2);
```

```
KeyCeer@LAPTOP-3O41Q4
   a: 1    b: 2
  n1: 2   n2: 2
n1++: 2 ++n2: 3
```

Orden de los operadores

Los operadores matemáticos siguen las reglas del PaPoMuDAS. (no existe un operador potencia en C)

Para operadores de la misma categoría se sigue el orden de izquierda a derecha

Tabla de prioridad para operadores

mayor prioridad								menor prioridad
()	++, --, -, !	*, /, %	+, -	<, >, <=, >=	==, !=	&&	||	=, +=, -=, *=, ….
Paréntesis, funciones	incremento, decremento y negación (operadores unarios)	multiplicación, división, módulo	suma y resta	Comparadores	comparadores de igualdad	Y	O	asignaciones

Estructuras condicionales en C

Permiten tomar decisiones en función de ciertas condiciones. Se utilizan para controlar el flujo del programa y ejecutar diferentes secciones de código según si una expresión es verdadera o falsa.

```
if(condición A) {
  … código …
}
```

```
if(condición A) {
  … código …
} else {
  … código …
}
```

```
if(condición A) {
  … código …
}else if(condición B) {
  … código …
}else if(condición C) {
  … código …
} else {
  … código …
}
```

Estructuras condicionales en C

Ejemplo de uso If-Else

```c
int a = 5;

if (a < 10) {
    printf("a es menor que 10\n");
} else if (a == 10) {
    printf("a es igual a 10\n");
} else {
    printf("a es mayor que 10\n");
}
```

If con operador lógico

```c
int a = 6;
int resto = a % 2;

if (a < 10 && resto == 0) {
    printf("a es menor que 10 y es par\n");
}
```

Switch-Case

Estructura de control en C que se utiliza para tomar decisiones múltiples basadas en el valor de una expresión. Proporciona una forma más organizada de manejar varias condiciones posibles en comparación con una serie de instrucciones “if-else if...”.

```
switch(selector) {
 case valor1:
   … código …
   break;
 case valor2:
   … código …
   break;
   ...
 case valorN:
   … código …
   break;
 default:
   … código …
   break;
}
```

Switch-Case

```c
int dia = 2;

switch (dia) {
    case 1:
        printf("Lunes\n");
        break;
    case 2:
        printf("Martes\n");
        break;
    case 3:
        printf("Miércoles\n");
        break;
    case 4:
        printf("Jueves\n");
        break;
    case 5:
        printf("Viernes\n");
        break;
    default:
        printf("Día no válido\n");
}
```

```c
int lados;

printf("Ingresa el número de lados de una figura geométrica: ");
scanf("%d", &lados);

switch (lados) {
    case 3:
        printf("Es un triángulo.\n");
        break;
    case 4:
        printf("Es un cuadrado.\n");
        break;
    case 5:
        printf("Es un pentágono.\n");
        break;
    case 6:
        printf("Es un hexágono.\n");
        break;
    case 7:
        printf("Es un heptágono.\n");
        break;
    case 8:
        printf("Es un octógono.\n");
        break;
    default:
        printf("No sé qué figura es.\n");
}
```

Ciclos en C

Son estructuras que permiten ejecutar un bloque de código repetidamente mientras se cumpla una condición determinada.

```
while(condición para continuar) {
            … código …
}
```

```
do {
            … código …
} while(condición para continuar)
```

```
for(valor inicial; condición término; incremento/decremento) {
            … código …
}
```

Ciclo While

Repite un bloque de código mientras la condición sea verdadera. Primero verifica la condición y luego ejecuta el código. Si la condición sigue siendo verdadera, se repite el proceso. Puede que el bloque nunca se ejecute si la condición inicialmente es falsa.

```c
int contador = 1;

while (contador <= 5) {
    printf("%d\n", contador);
    contador++;
}
```

Ciclo Do-While

Repite un bloque de código mientras la condición sea verdadera. A diferencia del ciclo **while**, el bloque de código se ejecuta al menos una vez, ya que la condición se verifica después de cada iteración.

```c
int numero;

do {
    printf("Ingresa un número (ingresa un negativo para salir): ");
    scanf("%d", &numero);

    printf("Número ingresado: %d\n", numero);
} while (numero >= 0);
```

Ciclo For

Repite un bloque de código un número específico de veces, controlado por una variable contador. Se compone de tres partes: inicialización, condición y operación de incremento/decremento.

```c
int i;

for (i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

Break - Continue

break y **continue** son palabras clave utilizadas en los ciclos en C para controlar el flujo de ejecución dentro de los ciclos.

break: Detiene por completo la ejecución del ciclo en el punto donde se encuentra.

continue: Salta a la siguiente iteración del ciclo, omitiendo el código restante en esa iteración.

Break - Continue

```c
int i = 0;

while(1){
    printf("%d\n", i);
    if(i == 10){
        break;
    }
    i++;
}
```

```c
int i = 0;

for (i = 0; i <= 10; i++) {
    if (i == 3 || i == 6) {
        continue;
    }
    printf("%d\n", i);
}
```

Arreglos en C

Son estructuras que permiten almacenar una colección de elementos del mismo tipo en una secuencia contigua en la memoria. Cada elemento en un arreglo se accede mediante un índice que comienza desde 0 para el primer elemento.

Total precipitación mensual en mm, año 2019. Estación Cerro Caracol, Concepción
Fuente: DGAC

	0	1	2	3	4	5	6	7	8	9	10	11
A =	18.8	1.7	16.9	5.9	214.3	336.2	144.4	68.6	79.1	42.7	18.9	0.0

Declaración de Arreglos en C

Se declaran especificando el tipo de datos de los elementos y la cantidad de elementos que contendrá el arreglo.

```c
int numeros[5];  // Declaración de un arreglo de enteros con 5 elementos
int n = 10;
float precios[n]; // Declaración de un arreglo de flotantes con n elementos
```

Inicialización de Arreglos en C

Los valores de los elementos del arreglo pueden ser inicializados al declarar el arreglo o después.

```c
// Inicialización al declarar
float flotantes[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
char palabra[4] = {"Hola"};
```

```c
// Inicialización después de declarar usando un ciclo
int enteros[5];
int i; // Variable de control del ciclo

for(i = 0; i < 5; i++){
    enteros[i] = i+1;
}
```

Acceso a los elementos del arreglo

Los elementos se acceden mediante su índice dentro de corchetes **[ ]**. Los índices comienzan desde 0 hasta tamaño - 1.

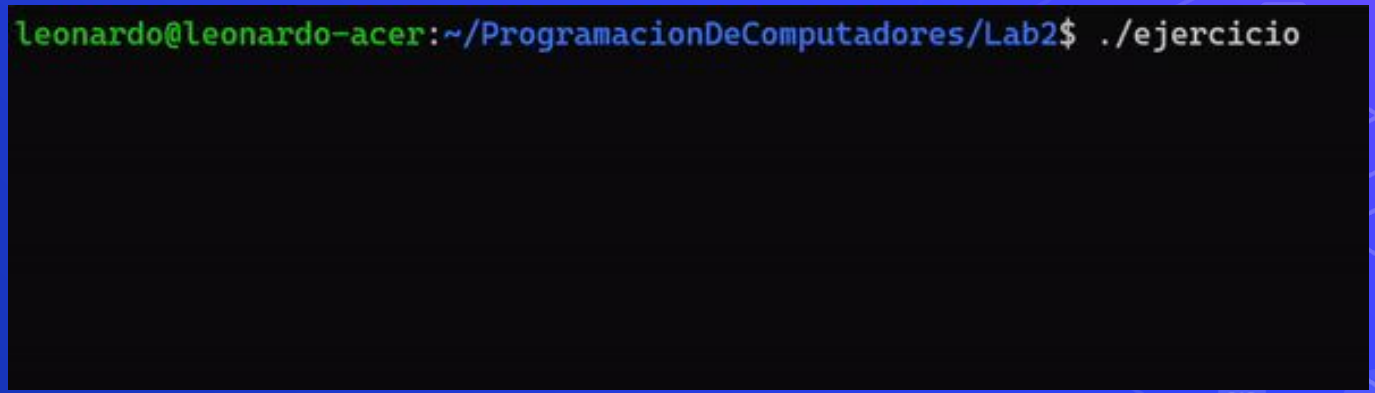
```
float flotantes[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
printf("El segundo elemento del arreglo es: %.1f\n", flotantes[1]);

flotantes[1] = 6.6;
printf("El nuevo valor del segundo elemento es: %.1f\n", flotantes[1]);
```

Ejercicio 1

Escribir un programa para recibir “n” cantidad de números enteros y guardar esos números en un arreglo. Luego debe mostrar en pantalla los números ingresados

El usuario debe escribir la cantidad “n” y luego los “n” números.



Ejercicio 2

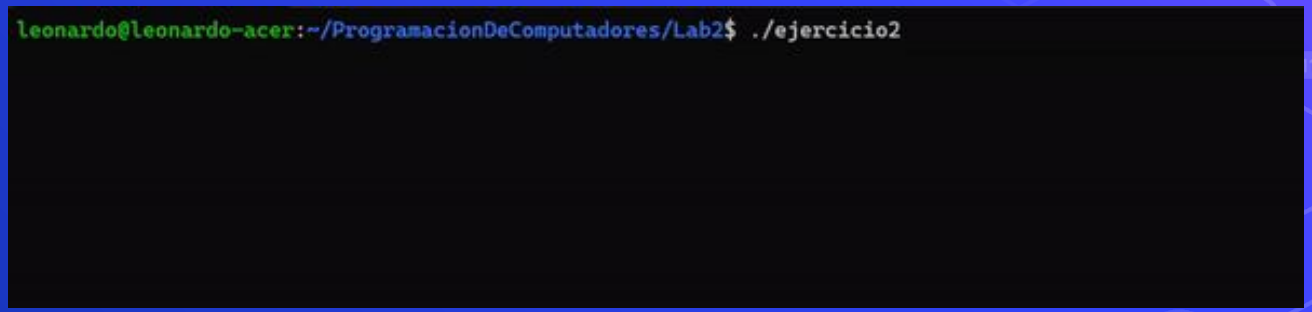
Modifique el programa anterior para que imprima en pantalla los números ingresados de manera inversa, desde el último ingresado hasta el primero.

Ejemplo:

Input:	**Output:**
`10 5 6 -9 8`	`8 -9 6 5 10`

Ejercicio 3

Modifique el ejercicio anterior para que se muestre en pantalla la suma de los números pares ingresados por el usuario.



Ejercicio 4

Modifique el ejercicio anterior para que se muestre en pantalla la suma de los números pares ingresados por el usuario y además la multiplicación de los números impares.

Ejemplo:

Input:

```
10 5 6 3 8 1
```

Output:

```
Suma de los pares: 24

Multiplicacion de los impares: 15
```