Universidad del Cauca

ISO 9001:2015 SC- CER 450832

IQNet: CO- SC-CER450832

Una Acreditación con Rostro Humano

Hacia una Universidad comprometida con la paz territorial

Universidad del Cauca

# Programación Imperativa en C

# Lab. 1. Recordando C

# Introducción

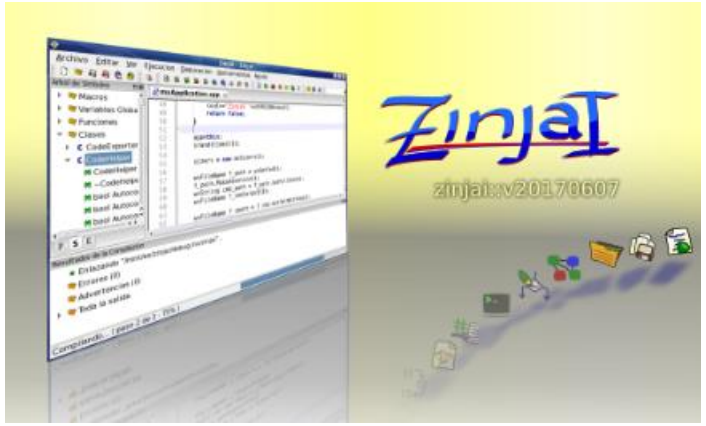
Universidad del Cauca

- C es un lenguaje de programación imperativo, ampliamente utilizado en desarrollo de sistemas y software.
- Permite el control total sobre la memoria y el hardware.
- Utiliza estructuras secuenciales, condicionales y repetitivas para ejecutar programas.
- Se basa en la modularidad con funciones y la manipulación directa de memoria.

# Zinjal IDE (C/C++)

Universidad del Cauca

- **Zinjal** es un IDE (entorno de desarrollo integrado) libre y gratuito para programar en C/C++.
- Pensado originalmente para ser utilizado por estudiantes de programación durante el aprendizaje, presenta una interfaz inicial muy sencilla, pero sin dejar de incluir funcionalidades avanzadas que permiten el desarrollo de proyectos tan complejos como el propio Zinjal.
- Zinjal es una excelente opción para desarrolladores que buscan un IDE rápido y eficiente.
- Su facilidad de uso y extensibilidad lo hacen una herramienta poderosa.
- **Ligero y rápido**, ideal para equipos con pocos recursos.
- **Personalizable**, permitiendo adaptar la experiencia según las necesidades del usuario.
- **Comunidad activa**, que contribuye con plugins y soporte.



# Característas Claves de C

Universidad del Cauca

- Uso de variables, constantes y operadores.
- Control de flujo con condicionales (if-else, switch).
- Bucles de repetición (for, while, do-while).
- Uso de funciones para modularidad.
- Manipulación de memoria con punteros.
- Uso de arreglos y estructuras para almacenar datos.

# Tipos de datos en C

Universidad del Cauca

| Tipo de Dato | Tamaño (bytes) | Descripción | Ejemplo |
|---|---|---|---|
| char | 1 | Carácter individual (signado) | char letra = 'A'; // -128 a 127 |
| unsigned char | 1 | Carácter individual sin signo | unsigned char letra = 65; // 0 a 255 |
| short int | 2 | Entero corto con signo | short int edad = 25; |
| unsigned short int | 2 | Entero corto sin signo | unsigned short int max_valor = 65535; |
| int | 4 | Entero estándar con signo | int numero = -100; |
| unsigned int | 4 | Entero sin signo | unsigned int cantidad = 200; |

**Tamaños de los tipos de datos en C en la plataforma Linux/Intel i686**

# Tipos de datos en C (Continuación)

| Tipo de Dato | Tamaño (bytes) | Descripción | Ejemplo |
|---|---|---|---|
| long int | 4 | Entero largo con signo | long int grande = 2147483647; |
| unsigned long int | 4 | Entero largo sin signo | unsigned long int max_long = 4294967295; |
| float | 4 | Número decimal de precisión simple | float pi = 3.1416; |
| double | 8 | Número decimal de precisión doble | double e = 2.718281828; |
| long double | 12 | Número decimal de precisión extendida | long double numero_grande = 1.23456789012345L; |

**Tamaños de los tipos de datos en C en la plataforma Linux/Intel i686**

# Entrada y salida de datos en C

Universidad del Cauca

- En el lenguaje de programación **C**, la entrada y salida de datos se maneja principalmente con las funciones de la biblioteca estándar **stdio.h**.
- La función **scanf** se usa para leer valores desde la entrada estándar (teclado) y almacenarlos en variables
  - scanf("formato", &variable);
  - "formato" define el tipo de dato que se va a leer (%d, %f, %c, etc.).
  - &variable usa el operador **"&"** (operador de dirección) para almacenar el valor ingresado en la memoria.

# Entrada y salida de datos en C

Universidad del Cauca

| Formato | Tipo de Dato | Ejemplo de scanf() | Ejemplo de printf() |
|---|---|---|---|
| %d | Entero (int) | scanf("%d", &num); | printf("Número: %d", num); |
| %f | Flotante (float) | scanf("%f", &num); | printf("Flotante: %f", num); |
| %lf | Doble precisión (double) | scanf("%lf", &num); | printf("Doble: %lf", num); |
| %c | Carácter (char) | scanf(" %c", &caracter);<br>// Nota: Espacio antes del %c | printf("Carácter: %c", caracter); |
| %s | Cadena de caracteres (char[]) | scanf("%s", cadena); | printf("Cadena: %s", cadena); |
| %x | Número hexadecimal (int) | scanf("%x", &num_hex); | printf("Hexadecimal: %x", num_hex); |
| %o | Número octal (int) | scanf("%o", &num_octal); | printf("Octal: %o", num_octal); |
| %p | Puntero (dirección de memoria) | scanf("%p", &puntero); | printf("Puntero: %p", puntero); |

# Ejercicio 1:

Universidad del Cauca

**Secuencialidad en C (entrada/salida, operaciones básicas).**

Escribir un programa que solicite al usuario dos números enteros. Realizar las cuatro operaciones básicas (+, -, *, /) y mostrar los resultados. ¿Cómo se manejan los datos de entrada y salida en C?

# Solución propuesta del ejercicio

Universidad del Cauca

```c
#include <stdio.h>
int main() {
    int primerNumero, segundoNumero;
    printf("Ingrese el primer número entero: ");
    scanf("%d", &primerNumero);
    printf("Ingrese el segundo número entero: ");
    scanf("%d", &segundoNumero);

    int suma = primerNumero + segundoNumero;
    int resta = primerNumero - segundoNumero;
    int multiplicacion = primerNumero * segundoNumero;
    double division = (segundoNumero != 0) ? (double) primerNumero / segundoNumero : 0;

    printf("\nResultados:\n");
    printf("Suma: %d\n", suma);
    printf("Resta: %d\n", resta);
    printf("Multiplicación: %d\n", multiplicacion);
    if (segundoNumero != 0)
        printf("División: %.2f\n", division);
    else
        printf("División: No se puede dividir por cero\n");
    return 0;
}
```

# Estructuras Condicionales en C

Universidad del Cauca

- Las **estructuras condicionales** en **C** permiten **tomar decisiones** en la ejecución del programa según ciertas condiciones.
- Las principales estructuras condicionales en C son:
  - if
  - if-else
  - if-else if-else
  - switch

# Estructuras Condicionales en C

Universidad del Cauca

| Estructura | Descripción | Ejemplo en C |
|---|---|---|
| if | Ejecuta un bloque si la condición es verdadera. | if (x > 0) { printf("Positivo"); } |
| if-else | Ejecuta un bloque si la condición es verdadera, otro si es falsa. | if (x > 0) { printf("Positivo"); } else { printf("No positivo"); } |
| if anidado | Un if dentro de otro if para evaluar múltiples condiciones. | if (x > 0) { printf("Positivo"); } else { if (x < 0) { printf("Negativo"); } } |
| if-else if-else | Evalúa múltiples condiciones en orden hasta encontrar la primera verdadera. | if (x >= 90) { printf("Excelente"); } else if (x >= 50) { printf("Aprobado"); } else { printf("Reprobado"); } |

# Estructuras Condicionales en C

Universidad del Cauca

| Estructura | Descripción | Ejemplo en C |
|---|---|---|
| if con && (AND) | Evalúa si ambas condiciones son verdaderas al mismo tiempo. | if (x > 0 && x < 10) { printf("Entre 1 y 9"); } |
| if con \|\| (OR) | Evalúa si al menos una de las condiciones es verdadera. | if (x < 0 \|\| x > 100) { printf("Fuera de rango"); } |
| if con ! (NOT) | Invierte el resultado de una condición (true -> false y viceversa). | if (!(x == 5)) { printf("No es cinco"); } |
| if con operadores combinados | Combina diferentes operadores lógicos para evaluar condiciones más complejas. | if ((x > 10 && x < 20) \|\| (y >= 5 && y <= 15)) { printf("Cumple condición"); } |
| switch | Evalúa una variable y ejecuta un bloque de código basado en su valor. | switch (opcion) { case 1: printf("Opción 1"); break; case 2: printf("Opción 2"); break; default: printf("Opción no válida"); } |

# Estructuras condicionales en C

Universidad del Cauca

Ejercicio 2: Uso de estructuras condicionales (if, switch).

Escribir un programa que solicite al usuario un número entero. Si el número es par, mostrar un mensaje indicándolo. Si es impar, mostrar otro mensaje. Usar el switch para determinar si el número es positivo, negativo o cero. ¿Cuál es la diferencia entre if-else y switch? ¿Cuándo es más eficiente usar switch en lugar de if-else?

# Solución propuesta del ejercicio 2

```c
#include <stdio.h>
int main() {
    int numero, estado;
    printf("Ingrese un número entero: ");
    scanf("%d", &numero);
    if (numero % 2 == 0) {
        printf("El número %d es par.\n", numero);
    } else {
        printf("El número %d es impar.\n", numero);
    }

    if (numero > 0) {
        estado = 1;
    } else if (numero < 0) {
        estado = -1;
    } else {
        estado = 0;
    }
    switch (estado) {
    case 1:
        printf("El número es positivo.\n");
        break;
    case -1:
        printf("El número es negativo.\n");
        break;
    case 0:
        printf("El número es cero.\n");
        break;
    default:
        printf("Error en la evaluación.\n");
    }

    return 0;
}
```

Universidad del Cauca

# Solución propuesta del ejercicio 2

Universidad del Cauca

```c
#include <stdio.h>
int main() {
    int numero;
    printf("Ingrese un número entero: ");
    scanf("%d", &numero);
    if (numero % 2 == 0) {
        printf("El número %d es par\n", numero);
    } else {
        printf("El número %d es impar\n", numero);
    }
    switch ((numero > 0) - (numero < 0)) {
    case 1:
        printf("El número es positivo.\n");
        break;
    case -1:
        printf("El número es negativo.\n");
        break;
    case 0:
        printf("El número es cero.\n");
        break;
    default:
        printf("Error en la evaluación.\n");
    }

    return 0;
}
```

# Estructuras Repetitivas (Bucles)

Universidad del Cauca

| Estructura | Descripción | Ejemplo en C |
|---|---|---|
| for | Se usa cuando se conoce el número exacto de iteraciones. | for (int i = 0; i < 10; i++) { printf("%d ", i); } |
| while | Se ejecuta mientras la condición sea verdadera. | int i = 0; while (i < 10) { printf("%d ", i); i++; } |
| do-while | Se ejecuta al menos una vez antes de verificar la condición. | int i = 0; do { printf("%d ", i); i++; } while (i < 10); |
| for anidado | Un bucle for dentro de otro para recorrer estructuras bidimensionales. | for (int i = 0; i < 3; i++) { for (int j = 0; j < 3; j++) { printf("(%d, %d) ", i, j); } } |
| while anidado | Un bucle while dentro de otro para evaluar múltiples condiciones. | int i = 0, j; while (i < 3) { j = 0; while (j < 3) { printf("(%d, %d) ", i, j); j++; } i++; } |
| Uso de break | Termina la ejecución del bucle antes de completar todas las iteraciones. | for (int i = 0; i < 10; i++) { if (i == 5) break; printf("%d ", i); } |
| Uso de continue | Salta la iteración actual y pasa a la siguiente. | for (int i = 0; i < 10; i++) { if (i == 5) continue; printf("%d ", i); } |

# Estructuras Repetitivas (Bucles)

Universidad del Cauca

**Ejercicio 3**

Escribir un programa en C que implemente diferentes estructuras repetitivas para resolver varias tareas. Primero, utilizar un bucle for para imprimir los primeros 15 números naturales en orden descendente. Luego, emplear un bucle while para mostrar los números impares hasta 30 en orden inverso. Posteriormente, con un bucle do-while, calcular el factorial de un número ingresado por el usuario, asegurando que el número sea válido. Además, solicitar al usuario un número n y calcular la suma de los primeros n números naturales usando un bucle for. Finalmente, explicar la diferencia entre for, while y do-while, destacando sus aplicaciones y cuándo es más eficiente utilizar cada uno.

# Estructuras Repetitivas (Bucles)

```c
#include <stdio.h>
int main() {
    int i, num, n, suma = 0, factorial = 1;
    printf("Primeros 15 números naturales en orden descendente:\n");
    for (i = 15; i >= 1; i--) {
        printf("%d ", i);
    }
    printf("\n\n");

    printf("Números impares hasta 30 en orden inverso:\n");
    i = 29;
    while (i >= 1) {
        printf("%d ", i);
        i -= 2;
    }
    printf("\n\n");

    do {
        printf("Ingrese un número positivo para calcular su factorial: ");
        scanf("%d", &num);
    } while (num < 0);

    i = num;
    do {
        factorial *= i;
        i--;
    } while (i > 0);

    printf("El factorial de %d es: %d\n\n", num, factorial);

    printf("Ingrese un número para calcular la suma de los primeros n números naturales: ");
    scanf("%d", &n);

    for (i = 1; i <= n; i++) {
        suma += i;
    }
    printf("La suma de los primeros %d números naturales es: %d\n", n, suma);

    return 0;
}
```

# Funciones en C

| Tipo de Función | Descripción | Ejemplo en C |
|---|---|---|
| Función sin retorno y sin parámetros | Ejecuta un bloque de código sin devolver un valor y sin recibir parámetros. | void mostrarMensaje() {<br>  printf("Hola, mundo!\n");<br>}<br>int main() {<br>  mostrarMensaje();<br>  return 0;<br>} |
| Función con retorno y sin parámetros | Ejecuta un bloque de código y devuelve un valor sin recibir parámetros. | int obtenerNumero() {<br>  return 42;<br>}<br>int main() {<br>  printf("Número: %d", obtenerNumero());<br>  return 0;<br>} |

Universidad del Cauca

# Funciones en C

| Tipo de Función | Descripción | Ejemplo en C |
|---|---|---|
| Función sin retorno pero con parámetros | Ejecuta un bloque de código, recibe parámetros pero no devuelve valores. | void imprimirNumero(int num) {<br>  printf("Número: %d\n", num);<br>}<br>int main() {<br>  imprimirNumero(10);<br>  return 0;<br>} |

Universidad del Cauca

# Funciones en C

| Tipo de Función | Descripción | Ejemplo en C |
|---|---|---|
| Función con retorno y con parámetros | Ejecuta un bloque de código, recibe parámetros y devuelve un valor. | int sumar(int a, int b) {<br>  return a + b;<br>}<br>int main() {<br>  int resultado = sumar(5, 3);<br>  printf("Suma: %d", resultado);<br>  return 0;<br>} |
| Paso de parámetros por valor | Los valores de los parámetros se copian en la función y no afectan la variable original. | void duplicar(int num) {<br>  num *= 2;<br>}<br>int main() {<br>  int x = 10;<br>  duplicar(x);<br>  printf("Valor de x: %d", x);<br>  return 0;<br>} |

# Funciones en C

| Tipo de Función | Descripción | Ejemplo en C |
|---|---|---|
| Paso de parámetros por referencia (punteros) | Se pasa la dirección de la variable, permitiendo modificar su valor dentro de la función. | void duplicarReferencia(int *num) {<br>*num *= 2;<br>}<br>int main() {<br>int x = 10;<br>duplicarReferencia(&x);<br>printf("Valor de x: %d", x);<br>return 0;<br>} |

# Modularidad con funciones

Universidad del Cauca

Ejercicio 4: Modularidad con funciones. Escribir un programa en C que solicite al usuario dos números enteros. Realizar las cuatro operaciones básicas (+, -, *, /) definiendo funciones y mostrar los resultados. ¿Por qué es importante dividir el código en funciones? ¿Cómo podríamos reutilizar las funciones definidas en otro programa?

# Solución propuesta del ejercicio 4

Universidad del Cauca

```c
#include <stdio.h>
int sumar(int a, int b) {
    return a + b;
}
int restar(int a, int b) {
    return a - b;
}
int multiplicar(int a, int b) {
    return a * b;
}
float dividir(int a, int b) {
    if (b == 0) {
        printf("Error: No se puede dividir por cero.\n");
        return 0;
    }
    return (float)a / b;
}
```

```c
int main() {
    int numUno, numDos;
    printf("Ingrese el primer número entero: ");
    scanf("%d", &numUno);
    printf("Ingrese el segundo número entero: ");
    scanf("%d", &numDos);

    printf("\nResultados de las operaciones:\n");
    printf("Suma: %d + %d = %d\n", numUno, numDos, sumar(numUno, numDos));
    printf("Resta: %d - %d = %d\n", numUno, numDos, restar(numUno, numDos));
    printf("Multiplicación: %d * %d = %d\n", numUno, numDos, multiplicar(numUno, numDos));
    printf("División: %d / %d = %.2f\n", numUno, numDos, dividir(numUno, numDos));

    return 0;
}
```

# Arreglos en C

Universidad del Cauca

- Un **arreglo en C** es una estructura de datos que permite almacenar **múltiples valores del mismo tipo** en una sola variable.
- Se accede a los elementos del arreglo mediante **índices**, comenzando desde 0
- Declaración de arreglo: tipo nombre[tamaño];
  - int numeros[5];

  Asignación directa:
  - int numeros[5] = {1, 2, 3, 4, 5};
  - int numeros[] = {10, 20, 30, 40};
  - int numeros[5] = {1, 2};

# Arreglos en C

Universidad del Cauca

- Se accede a los elementos por índice.

Ejemplo:

```
int numeros[5] = {1, 2, 3, 4, 5};
for (int i = 0; i < 5; i++) {
    printf("%d", numeros[i]);
}
```

# Arreglos en C

Universidad del Cauca

Ejercicio 5:

Escribir un programa en C que solicite al usuario 5 números enteros, los almacene en un arreglo y realice diversas operaciones sobre ellos. Primero, calcular el promedio de los números ingresados y mostrarlo en pantalla. Luego, determinar y mostrar el número mayor y el número menor dentro del arreglo. A continuación, ordenar los números en orden ascendente utilizando un algoritmo de ordenamiento y mostrar el arreglo ordenado. Finalmente, explicar cómo funcionan los arreglos en C, y cómo se almacenan los datos en memoria.

# ¡Gracias por su atención!

Universidad del Cauca

www.unicauca.edu.co