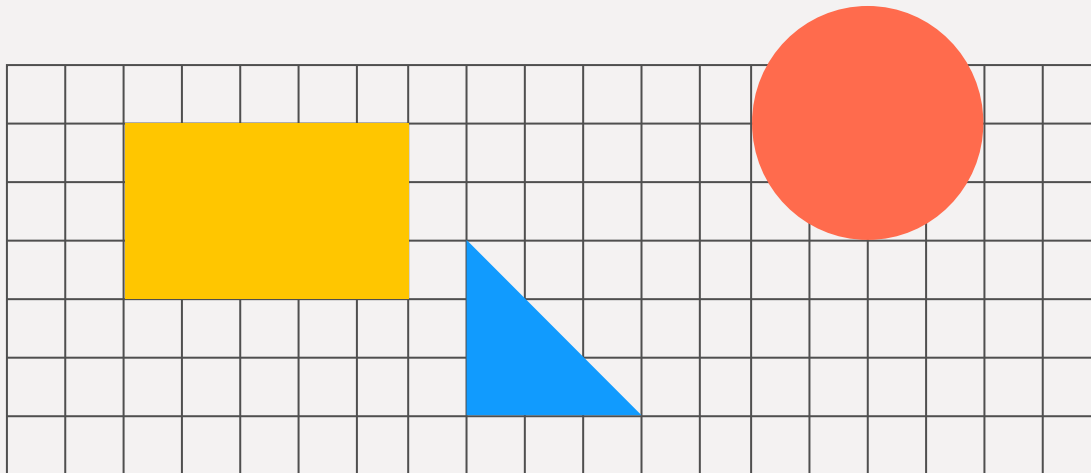


UNIDAD 1:

Introducción a C





Historia de C

1960

Lenguajes predecesores

Se usaban lenguajes como Fortran, COBOL y ALGOL, pero eran difíciles de usar para programación de sistemas.

En 1969, Ken Thompson creó el lenguaje B, basado en BCPL, para programar UNIX.

S

1972

Creación de C

Dennis Ritchie mejoró B y creó C, añadiendo tipado fuerte, mejor manejo de memoria y estructuras de control avanzadas.

Se usó para reescribir el núcleo de UNIX, haciéndolo más portable.

1978

K&R C

Brian Kernighan y Dennis Ritchie publicaron el libro "The C Programming Language", estableciendo el primer estándar informal de C, conocido como K&R C.

1989

ANSI C (C89)

Se estandarizó oficialmente como ANSI C (American National Standards Institute).

1990

ISO C

En 1990, el estándar pasó a ser ISO C (C90).
En 1999, se lanzó C99, agregando nuevas características como tipos enteros fijos (`int64_t`) y `inline`.
En 2011, salió C11, con mejoras en concurrencia y seguridad.
En 2018, apareció C18, con pequeñas correcciones al estándar anterior.

S HOY

Padre de los lenguajes

C sigue siendo la base de muchos sistemas operativos y lenguajes modernos como C++, Java, Python y Go.



Estructura de un Programa

► Estructura de un programa en C

```
1 // 1 Directivas del preprocesador
2 #include <stdio.h> // Librería estándar para entrada/salida
3 #include <math.h> // Librería para funciones matemáticas
4 // 2 Declaración de funciones (prototipos)
5 float calcularAreaCirculo(float radio);
6 // 3 Función principal (punto de entrada del programa)
7 int main()
8 {
9     // 4 Definición de variables
10    float radio, area;
11    // 5 Lógica del programa
12    printf("Ingrese el radio del círculo: ");
13    scanf("%f", &radio);
14    // Llamada a una función auxiliar
15    area = calcularAreaCirculo(radio);
16    printf("El área del círculo con radio %.2f es: %.2f\n", radio, area);
17    return 0; // 6 Fin del programa (código de salida)
18 }
19 // 7 Definición de funciones auxiliares
20 float calcularAreaCirculo(float radio)
21 {
22     return M_PI * radio * radio; // Fórmula del área de un círculo
23 }
```

¿Qué hace?

- ✓ **C tiene una estructura ordenada** que incluye directivas, funciones y una función `main()`.
- ✓ **Las variables deben declararse antes de usarlas** y pueden ser de distintos tipos.
- ✓ **El programa puede dividirse en funciones** para mejorar la organización y reutilización del código.
- ✓ **Siempre debe haber una función `main()`** porque es el punto de inicio del programa.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hola, mundo!\n");
6      return 0;
7  }
8
9
```


¿Qué hace?

- Compilamos el código: `gcc programa.c -o programa`
- Ejecutamos: `./programa`
- El sistema operativo crea un proceso para ejecutar el programa.
- El proceso imprime un mensaje en la consola.
- El proceso finaliza y libera sus recursos.



Procesos

► Procesos, un programa dentro de otro

```
1  #include <stdio.h>
2  #include <unistd.h> // Para usar sleep()
3  // Función que representa un proceso en ejecución
4  void procesoEjemplo()
5  {
6      printf("🟦 Proceso iniciado ... \n");
7      sleep(2); // Simula que el proceso está haciendo algo durante 2 segundos
8      printf("🟢 Proceso ejecutándose ... \n");
9      sleep(2);
10     printf("✅ Proceso finalizado.\n");
11 }
12 int main()
13 {
14     printf("💻 Iniciando programa ... \n");
15     // Llamamos a la función que simula un proceso
16     procesoEjemplo();
17     printf("🏁 Programa terminado.\n");
18     return 0;
19 }
20
```

¿Qué hace?

- El programa inicia e imprime "🖥️ Iniciando programa...".
- Llamamos a `procesoEjemplo()`, que representa un proceso en ejecución.
- Dentro de `procesoEjemplo()`:
 - Se imprime "🟦 Proceso iniciado...".
 - Se usa `sleep(2)`, lo que hace que el programa espere **2 segundos**.
 - Luego imprime "🟢 Proceso ejecutándose..." y espera otros **2 segundos**.
 - Finalmente, imprime "✅ Proceso finalizado."
- El control vuelve a `main()` y se imprime "🏁 Programa terminado."



Acciones y Estados

```
1  #include <stdio.h>
2  #include <unistd.h> // Para sleep()
3
4  void ejecutarProceso()
5  {
6      printf("🟦 Estado: NUEVO - Creando proceso ... \n");
7      sleep(1);
8      printf("🟡 Estado: LISTO - Esperando asignación de CPU ... \n");
9      sleep(1);
10     printf("🟢 Estado: EJECUCIÓN - El proceso está corriendo ... \n");
11     sleep(2); // Simula el tiempo de ejecución
12     printf("⌚ Estado: ESPERA - El proceso está esperando entrada del usuario ... \n");
13     char input;
14     printf("Presiona cualquier tecla y Enter para continuar: ");
15     scanf(" %c", &input); // Espera entrada del usuario
16     printf("✅ Estado: TERMINADO - El proceso ha finalizado.\n");
17 }
18
19 int main()
20 {
21     printf("🚀 Iniciando el programa ... \n");
22     ejecutarProceso();
23     printf("🏁 Programa finalizado.\n");
24     return 0;
25 }
```

¿Qué hace?

- El programa inicia en `main()` y llama a `ejecutarProceso()`.
- Estados del proceso en `ejecutarProceso()`:
 - **NUEVO**: Se indica que el proceso se está creando.
 - **LISTO**: El proceso espera ser ejecutado por el CPU.
 - **EJECUCIÓN**: El proceso está activo (simulado con `sleep(2)`).
 - **ESPERA**: Se pausa esperando entrada del usuario (`scanf`).
 - **TERMINADO**: El proceso finaliza.
- El programa finaliza y vuelve a `main()`, mostrando "🏁 Programa finalizado."

```
🚀 Iniciando el programa...
🟦 Estado: NUEVO - Creando proceso...
🟡 Estado: LISTO - Esperando asignación de CPU...
🟢 Estado: EJECUCIÓN - El proceso está corriendo...
🕒 Estado: ESPERA - El proceso está esperando entrada del usuario...
Presiona cualquier tecla y Enter para continuar: _
✅ Estado: TERMINADO - El proceso ha finalizado.
🏁 Programa finalizado.
```



Variables


```
1  #include <stdio.h>
2
3  int main()
4  {
5      // Declaración de variables
6      int edad = 25;      // Variable de tipo entero
7      float precio = 15.99; // Variable de tipo flotante
8      char letra = 'A';   // Variable de tipo carácter
9
10     // Mostramos los valores de las variables
11     printf("Edad: %d años\n", edad);
12     printf("Precio: $%.2f\n", precio);
13     printf("Letra: %c\n", letra);
14
15     // Modificamos la variable edad
16     edad = 30;
17     printf("Nueva edad: %d años\n", edad);
18
19     return 0;
20 }
21
```

¿Qué hace?


La variable `edad` cambió su valor de `25` a `30`, lo que demuestra que **una variable puede modificarse** en la ejecución.




Constantes

```
1  #include <stdio.h>
2
3  #define PI 3.1416 // Definimos una constante con #define
4
5  int main()
6  {
7      const float GRAVEDAD = 9.81; // Definimos una constante con const
8      float radio = 5.0;
9
10     // Calculamos el área de un círculo con PI
11     float area = PI * radio * radio;
12
13     printf("Radio: %.2f\n", radio);
14     printf("Área del círculo: %.2f\n", area);
15     printf("Valor de la gravedad: %.2f m/s^2\n", GRAVEDAD);
16
17     // GRAVEDAD = 10; ❌ Esto daría error porque es una constante
18     return 0;
19 }
20
```

¿Qué hace?

 Una **constante** es un valor que **no cambia** durante la ejecución del programa.

 Se usa la palabra clave **const** o la directiva **#define** para definirlas.

Diferencias entre Variables y Constantes

- ✓ Las **variables** permiten almacenar y modificar datos en la memoria.
- ✓ Las **constantes** almacenan valores fijos que no pueden cambiar.
- ✓ Se pueden definir **constantes** con **const** o **#define**, pero **#define** no tiene tipo de dato.

Característica	Variable	Constante
Valor	Puede cambiar en la ejecución	No cambia una vez definida
Declaración	<code>int edad = 25;</code>	<code>const float PI = 3.1416;</code> o <code>#define PI 3.1416</code>
Modificable	✓ Sí	✗ No
Ejemplo	<code>edad = 30;</code>	<code>PI = 3.5;</code> // ✗ ERROR



Operaciones básicas de salida

```
#include <stdio.h>

int main()
{
    int edad = 25;
    float precio = 10.5;
    char letra = 'A';
    char nombre[] = "Carlos";

    printf("Hola, mi nombre es %s.\n", nombre);
    printf("Tengo %d años.\n", edad);
    printf("El precio es $%.2f\n", precio);
    printf("La primera letra del abecedario es %c.\n", letra);

    return 0;
}
```


Especificadores de Formato en printf()

Especificador	Tipo de Dato	Ejemplo
<code>%d o %i</code>	Enteros	<code>int edad = 25; printf("%d", edad);</code>
<code>%f</code>	Flotantes (decimales)	<code>float pi = 3.1416; printf("%f", pi);</code>
<code>%.2f</code>	Flotantes con 2 decimales	<code>printf("%.2f", 3.1416);</code> → 3.14
<code>%c</code>	Caracteres	<code>char letra = 'A'; printf("%c", letra);</code>
<code>%s</code>	Cadenas de texto	<code>char nombre[] = "Juan"; printf("%s", nombre);</code>

Salto de Línea y Caracteres Especiales

Carácter Especial	Función
<code>\n</code>	Salto de línea
<code>\t</code>	Tabulación (espacio grande)
<code>\\</code>	Imprime una barra invertida (<code>\</code>)
<code>\"</code>	Imprime comillas dobles (<code>"</code>)