C es un lenguaje de programación imperativo, eso significa que debemos describir completamente lo que debe hacer.

instrucciones de distintos tipos:

1. Asignaciones Guarda un valor en la memoria temporal
   Por ejemplo, si estamos haciendo un cálculo con un valor determinado podemos mantenerlo en la memoria mientras lo hacemos.

2. Condicionales En función de una determinada condición podemos ejecutar una serie de instrucciones u otras.
Por ejemplo, si desarrollamos una calculadora y queremos hacer una división en que el dividendo sea 0, podemos hacer que el programa se niegue, y decida que serie de pasos seguir.

3. Bucles

El programa repite una serie de instrucciones un número de veces o hasta que una condición se cumpla.
Por ejemplo, al hacer un sumatorio podemos guardar en una variable temporal mediante una asignación el acumulador y luego hacer un bucle para que se repitan las interacciones en orden.

4. Llamadas

Una función invoca a otra función.
< C es un lenguaje de programación procedimental, lo que quiere decir que nuestro programa va a estar dividido en regiones de código llamadas funciones o procedimientos según el tipo que sea >

ENTRADA → FUNCIÓN → SALIDA

Un símil matemático sería interpretar el cosα como una función, en la que introduces un valor, como puede ser el 0, y te devuelve otro, que sería 1 ($\cos 0 = 1$)

En C todo el código se encontrará dentro de la función, nada fuera.

Tipos de datos:

1. Números enteros

Se van a representar con la palabra clave `int`

2. Números decimales o flotantes

Se van a llamar `float` y se separan en una categoría distinta a la de los enteros porque el procesador los trata diferente.

Al escribir números decimales ponemos punto (.) en lugar de comas.

3. Caracteres

Definidos por `char`. Abarcan desde letras, números, símbolos y van entre comillas simples.

Las cadenas de caracteres se encontrarán definidas entre comillas normales ("hola mundo").

4. Void

Sirve para construir funciones que no nos devuelvan nada.

¿Cómo construyo funciones en C?

- Identificar cómo se va a llamar la función
- Identificar qué parámetros de entrada va a tener.
- Identificar el tipo de datos de retorno que vamos a tener.

EJEMPLO

```
int main ( ) {
    <tab> return 0;
}
```

- `int` → lo que nos devuelvas
- `main` → función
- `( )` → Parámetros de esa función (En este caso no hay parámetros estipulados)
- `{` → cuerpo de la función
- `return 0;` → return 0 indica que la respuesta que nos va a dar el programa siempre va a ser 0, ninguna otra cosa.

Este ejemplo se suele utilizar para saber si toda la ejecución se ha realizado correctamente.
En este caso, que nos muestre 0 confirma lo anterior.

Para imprimir un mensaje en pantalla

Introduciremos una llamada a otra función, que sería `printf` en este caso. Deberemos introducirle un parámetro, aquí sería la lista de caracteres que mostraremos en pantalla.

```
> int main() {
>     printf("Hola, mundo.");
>
>     return 0;
> }
```

Al final de cada instrucción hay que escribir ;

Al final nos queda un programa compuesto de dos instrucciones

Realmente este código no está del todo correcto, ya que carece de archivos de cabecera.

Los archivos de cabecera contienen definiciones de funciones, ahí especifica, por ejemplo, qué debe hacer printf.

¿Cómo hago uno?

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

↳ Este es el archivo donde se encuentran todas las definiciones

Ahora ya podemos compilar y ejecutar sin problemas.