

UNIVERSIDAD NACIONAL DE INGENIERÍA

Introducción a la Computación

Qué es computar

Algoritmos, terminal y primer programa en C

Gerson Garrido

Semana 1

Hoy

1. Cómo funciona el curso
2. Algoritmos: pasos precisos, finitos y sin ambigüedad
3. Pseudocódigo y diagramas de flujo. Seguir un algoritmo a mano
4. Niveles de lenguaje y compiladores
5. La terminal, desde cero
6. El primer programa en C y sus errores

> **Lectura**
> Los apuntes de la semana 1 son el texto del curso. No hay libro.

SECCIÓN 01

El curso

01

Cuatro partes

Fundamentos

algoritmos, variables, bits, terminal, Git

sem. 1–3

Cuatro partes

Fundamentos

algoritmos, variables, bits, terminal, Git

sem. 1–3

Programación en C

decisiones, ciclos, funciones, pruebas

sem. 4–7

Cuatro partes

Fundamentos

algoritmos, variables, bits, terminal, Git

sem. 1–3

Programación en C

decisiones, ciclos, funciones, pruebas

sem. 4–7

Memoria y datos

arreglos, cadenas, `struct`, archivos, eficiencia

sem. 9–13

Cuatro partes

Fundamentos

algoritmos, variables, bits, terminal, Git

sem. 1–3

Programación en C

decisiones, ciclos, funciones, pruebas

sem. 4–7

Memoria y datos

arreglos, cadenas, `struct`, archivos, eficiencia

sem. 9–13

Computación moderna

Python, internet, inteligencia artificial

sem. 14–15

Cómo funciona el curso

- **Teoría** (2 h) y **laboratorio** (2 h) cada semana.
- Cada algoritmo: primero pseudocódigo, después C.
- Trabajo individual: cada uno escribe y explica su código.

Prácticas y exámenes

Prácticas: semanas 4, 7, 11 y 14
Parcial: semana 8
Final: semana 16

SECCIÓN 02

Algoritmos

02

Qué significa computar

- Seguir pasos, uno tras otro.

Qué significa computar

- Seguir pasos, uno tras otro.
- Sin entender lo que se hace.

Qué significa computar

- Seguir pasos, uno tras otro.
- Sin entender lo que se hace.
- Sin agregar nada de cosecha propia.

Qué significa computar

- Seguir pasos, uno tras otro.
- Sin entender lo que se hace.
- Sin agregar nada de cosecha propia.

La consecuencia

Una computadora solo hace lo que alguien le explicó, sin dejar detalles a su criterio. De eso trata este curso.

Algoritmo *(algorithm)*

Definición

Secuencia finita de pasos precisos que resuelve un problema.

Condición	Significa	Ejemplo que la rompe
Preciso	Cada paso dice qué hacer	“Agrega sal al gusto”

Algoritmo *(algorithm)*

Definición

Secuencia finita de pasos precisos que resuelve un problema.

Condición	Significa	Ejemplo que la rompe
Preciso	Cada paso dice qué hacer	“Agrega sal al gusto”
Sin ambigüedad	Dos personas hacen lo mismo	“Dobla por la esquina de siempre”

Algoritmo *(algorithm)*

Definición

Secuencia finita de pasos precisos que resuelve un problema.

Condición	Significa	Ejemplo que la rompe
Preciso	Cada paso dice qué hacer	“Agrega sal al gusto”
Sin ambigüedad	Dos personas hacen lo mismo	“Dobla por la esquina de siempre”
Finito	Termina	“Lava, enjuaga, repite”

La trampa del sentido común

- “Dale su vuelto” funciona con un cajero.

La trampa del sentido común

- “Dale su vuelto” funciona con un cajero.
- El cajero ya sabe cómo se hace.

La trampa del sentido común

- “Dale su vuelto” funciona con un cajero.
- El cajero ya sabe cómo se hace.
- No le dice nada a quien no lo sabe.

La trampa del sentido común

- “Dale su vuelto” funciona con un cajero.
- El cajero ya sabe cómo se hace.
- No le dice nada a quien no lo sabe.

Error común

Escribir pasos que una persona entiende por contexto, pero que no dicen exactamente qué hacer.

La trampa del sentido común

- “Dale su vuelto” funciona con un cajero.
- El cajero ya sabe cómo se hace.
- No le dice nada a quien no lo sabe.

Error común

Escribir pasos que una persona entiende por contexto, pero que no dicen exactamente qué hacer.

Para probarlo: léanlo como una máquina obediente y sin sentido común. Si tienen que adivinar, está mal.

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80		

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80
S/ 2.80	S/ 2	S/ 0.80

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80
S/ 2.80	S/ 2	S/ 0.80
S/ 0.80	S/ 0.50	S/ 0.30

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80
S/ 2.80	S/ 2	S/ 0.80
S/ 0.80	S/ 0.50	S/ 0.30
S/ 0.30	S/ 0.20	S/ 0.10

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80
S/ 2.80	S/ 2	S/ 0.80
S/ 0.80	S/ 0.50	S/ 0.30
S/ 0.30	S/ 0.20	S/ 0.10
S/ 0.10	S/ 0.10	S/ 0.00

Un problema: dar vuelto

Vuelto de **S/ 7.80** con la menor cantidad de monedas. Suponemos monedas de S/ 5, 2, 1, 0.50, 0.20 y 0.10.

Faltan	La más grande que cabe	Quedan
S/ 7.80	S/ 5	S/ 2.80
S/ 2.80	S/ 2	S/ 0.80
S/ 0.80	S/ 0.50	S/ 0.30
S/ 0.30	S/ 0.20	S/ 0.10
S/ 0.10	S/ 0.10	S/ 0.00

Contaremos en céntimos (780): todo entero y las cuentas son exactas.

SECCIÓN 03

Pseudocódigo

03

Pseudocódigo *(pseudocode)*

- Lo leen personas. No lo ejecuta ninguna máquina.
- Una **variable** es un nombre para un lugar con un valor.
- ← guarda un valor: primero se calcula la derecha.
- Lo de adentro va con sangría.

monedas_de_cinco

```
1  algoritmo monedas_de_cinco(vuelto)
2      monedas ← 0
3      mientras vuelto ≥ 500 hacer
4          vuelto ← vuelto - 500
5          monedas ← monedas + 1
6      fin mientras
7      escribir monedas
8  fin algoritmo
```

Asignar no es igualar

`monedas ← monedas + 1` no es una ecuación: es “aumenta `monedas` en 1”.

Repetir y decidir

`mientras`

1. ¿La condición es cierta?
2. Sí: ejecuta lo de adentro y vuelve a 1.
3. No: sigue después de `fin mientras`.

Puede no ejecutarse ni una vez.

`si`: se ejecuta una sola rama.

mayor_de_dos

```
1  algoritmo mayor_de_dos(a, b)
2      si a > b entonces
3          escribir a
4      sino
5          escribir b
6      fin si
7  fin algoritmo
```

El vuelto: seis bloques iguales

dar_vuelto

```
1  algoritmo dar_vuelto(vuelto)
2      monedas ← 0
3      mientras vuelto ≥ 500 hacer
4          escribir 500
5          vuelto ← vuelto - 500
6          monedas ← monedas + 1
7      fin mientras
8      mientras vuelto ≥ 200 hacer
9          escribir 200
10         vuelto ← vuelto - 200
11         monedas ← monedas + 1
12     fin mientras
```

- Un bloque por cada moneda: 500, 200, 100, 50, 20 y 10.
- Al final: `escribir monedas` y `fin algoritmo`.
- El completo está en los apuntes.

Seguir a mano

Ejecutarlo ustedes, con una tabla: una columna por variable.

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	
8	¿30 $\geq$ 20?	sí	10	4	20

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	
8	¿30 $\geq$ 20?	sí	10	4	20
9	¿10 $\geq$ 20?	no	10	4	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	
8	¿30 $\geq$ 20?	sí	10	4	20
9	¿10 $\geq$ 20?	no	10	4	
10	¿10 $\geq$ 10?	sí	0	5	10

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	
8	¿30 $\geq$ 20?	sí	10	4	20
9	¿10 $\geq$ 20?	no	10	4	
10	¿10 $\geq$ 10?	sí	0	5	10
11	¿0 $\geq$ 10?	no	0	5	

Seguir `dar_vuelto` con 780

Paso	Qué se ejecuta	¿Cierto?	`vuelto`	`monedas`	Escribe
0	Inicio		780	0	
1	¿780 $\geq$ 500?	sí	280	1	500
2	¿280 $\geq$ 500?	no	280	1	
3	¿280 $\geq$ 200?	sí	80	2	200
4	¿80 $\geq$ 200?	no	80	2	
5	¿80 $\geq$ 100?	no	80	2	
6	¿80 $\geq$ 50?	sí	30	3	50
7	¿30 $\geq$ 50?	no	30	3	
8	¿30 $\geq$ 20?	sí	10	4	20
9	¿10 $\geq$ 20?	no	10	4	
10	¿10 $\geq$ 10?	sí	0	5	10
11	¿0 $\geq$ 10?	no	0	5	
12	`escribir monedas`		0	5	5

Tres cosas que pueden salir mal

Ciclo infinito

Sin `vuelto` $\leftarrow$ `vuelto - 500`, ¿780 $\geq$ 500? siempre dice sí. Nunca termina.

Tres cosas que pueden salir mal

Ciclo infinito

Sin `vuelto` $\leftarrow$ `vuelto - 500`, ¿780 $\geq$ 500? siempre dice sí. Nunca termina.

Datos no válidos

Con 785, `vuelto` termina en 5 y nadie avisa. El algoritmo supone un múltiplo de 10 no negativo.

Tres cosas que pueden salir mal

Ciclo infinito

Sin `vuelto` $\leftarrow$ `vuelto - 500`, ¿780 $\geq$ 500? siempre dice sí. Nunca termina.

Datos no válidos

Con 785, `vuelto` termina en 5 y nadie avisa. El algoritmo supone un múltiplo de 10 no negativo.

¿La menor cantidad?

Con monedas de 1, 3 y 4 para dar 6, el método da 4, 1, 1. Pero 3, 3 usa menos.

Tres cosas que pueden salir mal

Ciclo infinito

Sin `vuelto` $\leftarrow$ `vuelto - 500`, ¿780 $\geq$ 500? siempre dice sí. Nunca termina.

Datos no válidos

Con 785, `vuelto` termina en 5 y nadie avisa. El algoritmo supone un múltiplo de 10 no negativo.

¿La menor cantidad?

Con monedas de 1, 3 y 4 para dar 6, el método da 4, 1, 1. Pero 3, 3 usa menos.

Con las monedas de este problema sí da la menor cantidad: lo comprobamos con un programa para cada monto de 10 en 10 céntimos hasta S/ 100.

Tres cosas que pueden salir mal

Ciclo infinito

Sin `vuelto` $\leftarrow$ `vuelto - 500`, ¿780 $\geq$ 500? siempre dice sí. Nunca termina.

Datos no válidos

Con 785, `vuelto` termina en 5 y nadie avisa. El algoritmo supone un múltiplo de 10 no negativo.

¿La menor cantidad?

Con monedas de 1, 3 y 4 para dar 6, el método da 4, 1, 1. Pero 3, 3 usa menos.

Con las monedas de este problema sí da la menor cantidad: lo comprobamos con un programa para cada monto de 10 en 10 céntimos hasta S/ 100.

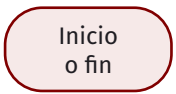
Un ejemplo que sale bien no prueba que sea correcto. Uno que sale mal prueba que no lo es.

SECCIÓN 04

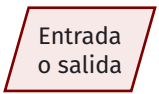
Diagramas de flujo

04

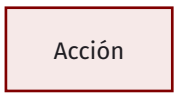
Las formas



`algoritmo`



`leer, escribir`



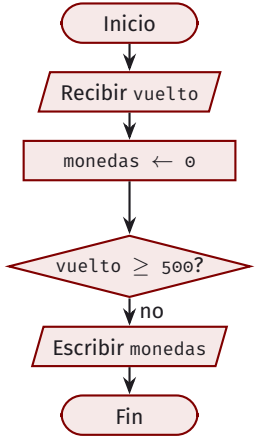
`monedas ← 0`



`si, mientras`

Las flechas indican qué paso sigue. Un rombo tiene dos salidas: *sí* y *no*.

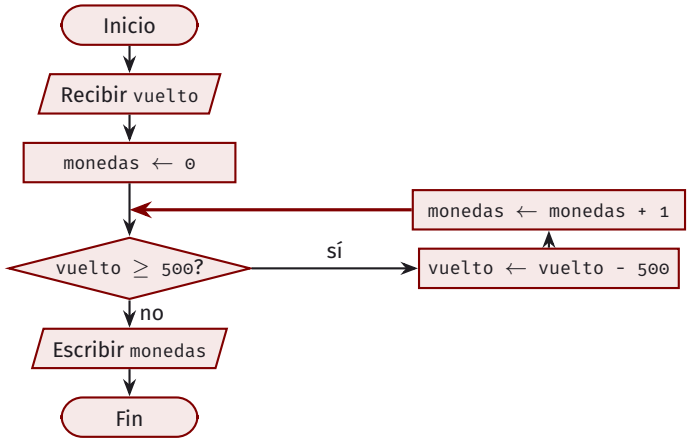
`monedas_de_cinco` como diagrama



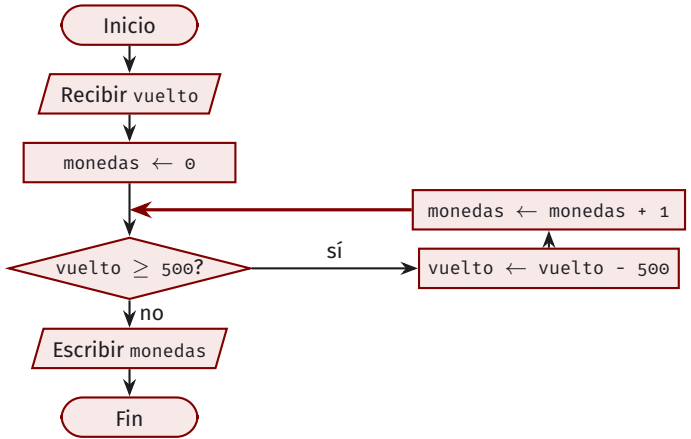

```
monedas ← monedas + 1
```

```
vuelto ← vuelto - 500
```

`monedas_de_cinco` como diagrama



`monedas_de_cinco` como diagrama



La flecha granate va hacia atrás: eso es el ciclo. Con `vuelto` = 1300, escribe 2.

SECCIÓN 05

De las ideas a la máquina

05

Niveles de lenguaje

Nivel	Ejemplo	Quién lo usa
Pseudocódigo	`escribir "Hola, mundo"`	Personas, para pensar

Niveles de lenguaje

Nivel	Ejemplo	Quién lo usa
Pseudocódigo	`escribir "Hola, mundo"`	Personas, para pensar
Alto nivel (C)	`printf("Hola, mundo\n");`	Personas, para programar

Niveles de lenguaje

Nivel	Ejemplo	Quién lo usa
Pseudocódigo	`escribir "Hola, mundo"`	Personas, para pensar
Alto nivel (C)	`printf("Hola, mundo\n");`	Personas, para programar
Bajo nivel (ensamblador)	`bl puts`	Personas, en casos especiales

Niveles de lenguaje

Nivel	Ejemplo	Quién lo usa
Pseudocódigo	`escribir "Hola, mundo"`	Personas, para pensar
Alto nivel (C)	`printf("Hola, mundo\n");`	Personas, para programar
Bajo nivel (ensamblador)	`bl puts`	Personas, en casos especiales
Código máquina	Números	La CPU

Niveles de lenguaje

Nivel	Ejemplo	Quién lo usa
Pseudocódigo	`escribir "Hola, mundo"`	Personas, para pensar
Alto nivel (C)	`printf("Hola, mundo\n");`	Personas, para programar
Bajo nivel (ensamblador)	`bl puts`	Personas, en casos especiales
Código máquina	Números	La CPU

El nivel es relativo

C es de alto nivel comparado con el código máquina, y de más bajo nivel que Python. Cada familia de procesadores tiene su propio código máquina.

Compilador *(compiler)*



Compilador *(compiler)*



Compilador *(compiler)*



- `hola.c`: texto para personas. Es lo que escriben y conservan.

Compilador *(compiler)*



- `hola.c`: texto para personas. Es lo que escriben y conservan.
- `hola`: código máquina. Es lo que ejecuta la computadora, y se puede regenerar.

Compilador *(compiler)*



- `hola.c`: texto para personas. Es lo que escriben y conservan.
- `hola`: código máquina. Es lo que ejecuta la computadora, y se puede regenerar.
- Si cambian `hola.c`, `hola` no se entera: hay que compilar de nuevo.

SECCIÓN 06

La terminal

06



Terminal, shell, prompt

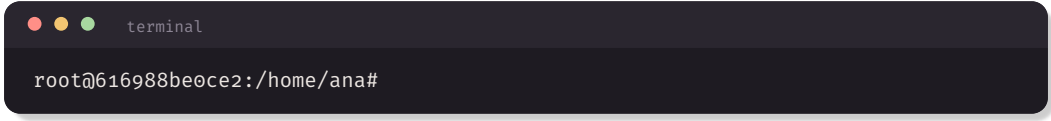
Terminal La ventana donde se escribe y se leen las respuestas

Terminal, shell, prompt

Terminal	La ventana donde se escribe y se leen las respuestas
Shell	El programa que lee lo que escriben y lo ejecuta: `bash`

Terminal, shell, prompt

Terminal	La ventana donde se escribe y se leen las respuestas
Shell	El programa que lee lo que escriben y lo ejecuta: `bash`
Prompt	El texto que indica que el shell está listo para recibir una orden



Terminal, shell, prompt

Terminal	La ventana donde se escribe y se leen las respuestas
Shell	El programa que lee lo que escriben y lo ejecuta: `bash`
Prompt	El texto que indica que el shell está listo para recibir una orden

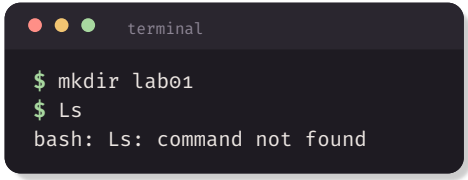
terminal

```
root@616988be0ce2:/home/ana#
```

Usuario, máquina y carpeta actual. El de ustedes será distinto. En el curso lo escribimos como `$`: no se teclea, y las demás líneas son la respuesta.

Comandos y argumentos

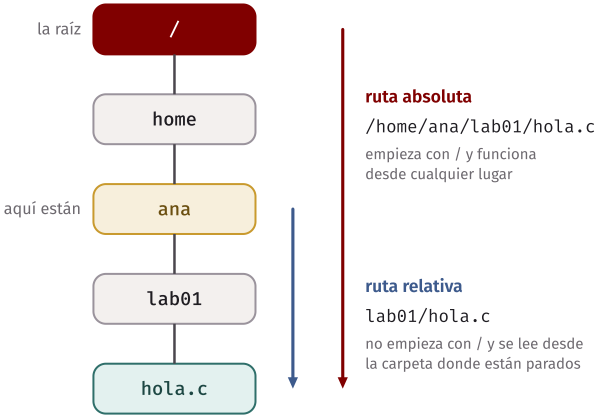
- `mkdir` es el comando; `lab01`, su **argumento**.
- Las **opciones** empiezan con guion: `ls -l`.
- El espacio separa, y las mayúsculas importan.
- Nada ocurre hasta presionar **Enter**.



Los mensajes ayudan

Vienen en inglés. Aprendan a leerlos sin miedo: *command not found* es "comando no encontrado".

Carpetas y rutas



Tres atajos en las rutas

`.`	La carpeta actual
`..`	La carpeta de arriba
`~`	Su carpeta personal (en el entorno: `/root`)

Los cinco comandos de hoy

Comando	Viene de	Qué hace
`pwd`	print working directory	Escribe la ruta de la carpeta actual
`ls`	list	Lista lo que hay en la carpeta actual
`cd ruta`	change directory	Cambia la carpeta actual
`mkdir nombre`	make directory	Crea una carpeta
`cat archivo`	concatenate	Muestra el contenido de un archivo

Una sesión

terminal

```
$ pwd
/home/ana
$ ls
$ mkdir lab01
$ ls
lab01
$ cd lab01
$ pwd
/home/ana/lab01
$ cd ..
$ pwd
/home/ana
```

- `ls` no escribe nada: la carpeta está vacía.

Una sesión

terminal

```
$ pwd
/home/ana
$ ls
$ mkdir lab01
$ ls
lab01
$ cd lab01
$ pwd
/home/ana/lab01
$ cd ..
$ pwd
/home/ana
```

- `ls` no escribe nada: la carpeta está vacía.
- `mkdir` no escribe nada: funcionó.

Una sesión

terminal

```
$ pwd
/home/ana
$ ls
$ mkdir lab01
$ ls
lab01
$ cd lab01
$ pwd
/home/ana/lab01
$ cd ..
$ pwd
/home/ana
```

- `ls` no escribe nada: la carpeta está vacía.
- `mkdir` no escribe nada: funcionó.
- `cd` no escribe nada, pero la carpeta actual cambió.

Una sesión

terminal

```
$ pwd
/home/ana
$ ls
$ mkdir lab01
$ ls
lab01
$ cd lab01
$ pwd
/home/ana/lab01
$ cd ..
$ pwd
/home/ana
```

- `ls` no escribe nada: la carpeta está vacía.
- `mkdir` no escribe nada: funcionó.
- `cd` no escribe nada, pero la carpeta actual cambió.

El silencio es buena noticia

Si un comando funciona y no tiene nada que mostrar, no dice nada. Si falla, se queja.

Subir niveles y equivocarse

Estando en `/home/ana/labo1`:

terminal

```
$ cd ../..
$ pwd
/home
$ cd ~
$ pwd
/home/ana
```

Equivocarse no rompe nada:

terminal

```
$ cd labo1
bash: cd: labo1: No such file or directory
$ cd mi carpeta
bash: cd: too many arguments
```

Revisen con `ls` qué hay y con `pwd` dónde están. Sin espacios ni tildes en los nombres.

Tres ayudas

Tab

Completa nombres. Escriben menos y no se equivocan.

Flecha arriba

Trae el comando anterior para repetirlo o editarlo.

Ctrl+C

Corta un programa colgado. Ctrl+D: “no hay más datos”.

Cuidado

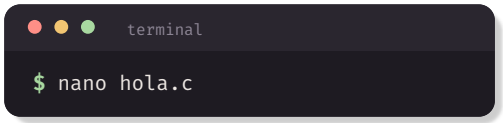
`rm archivo` borra un archivo. La terminal no tiene papelera.

Escribir un archivo

El editor del entorno

- Codespaces: el panel de arriba, o `code hola.c`.
- Docker: el editor de su laptop; la carpeta es la misma.
- Guardar: **Ctrl+S**.

nano, en la terminal



Ctrl+O y Enter: guarda. Ctrl+X: sale.

Error común

Para `gcc` solo existe lo que está **guardado**. Si el error sigue igual, revisen con `cat hola.c`.

SECCIÓN 07

El primer programa

07



hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.
- `main`: aquí empieza todo programa. `int`: devuelve un entero. `void`: no recibe datos.

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.
- `main`: aquí empieza todo programa. `int`: devuelve un entero. `void`: no recibe datos.
- `printf`: escribe el texto. `\n` es un salto de línea.

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.
- `main`: aquí empieza todo programa. `int`: devuelve un entero. `void`: no recibe datos.
- `printf`: escribe el texto. `\n` es un salto de línea.
- `;` termina cada instrucción.

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.
- `main`: aquí empieza todo programa. `int`: devuelve un entero. `void`: no recibe datos.
- `printf`: escribe el texto. `\n` es un salto de línea.
- `;` termina cada instrucción.
- `return 0`: “todo salió bien”.

hola.c

hola.c

```c
#include <stdio.h>

int main(void) {
    printf("Hola, mundo\n");
    return 0;
}
```

- `#include <stdio.h>`: presenta `printf` al compilador.
- `main`: aquí empieza todo programa. `int`: devuelve un entero. `void`: no recibe datos.
- `printf`: escribe el texto. `\n` es un salto de línea.
- `;` termina cada instrucción.
- `return 0`: “todo salió bien”.

Las instrucciones se ejecutan en orden, de arriba hacia abajo.

Escribir, compilar, ejecutar

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
$ ls
hola  hola.c
$ ./hola
Hola, mundo
```

`gcc`	El compilador
`-Wall -Wextra`	Pide que avise de muchas cosas sospechosas
`-std=c17`	Usa la versión C17 del lenguaje
`hola.c`	El archivo a traducir
`-o hola`	El nombre del ejecutable que debe crear

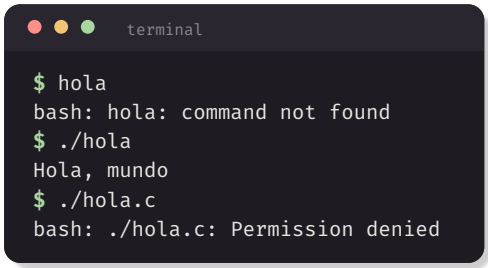
¿Por qué `./hola`?

terminal

```
$ hola
bash: hola: command not found
$ ./hola
Hola, mundo
$ ./hola.c
bash: ./hola.c: Permission denied
```

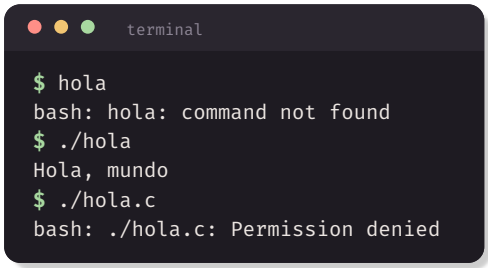
- Una palabra sola se busca en una lista fija de carpetas del sistema. La actual no está.

¿Por qué `./hola`?



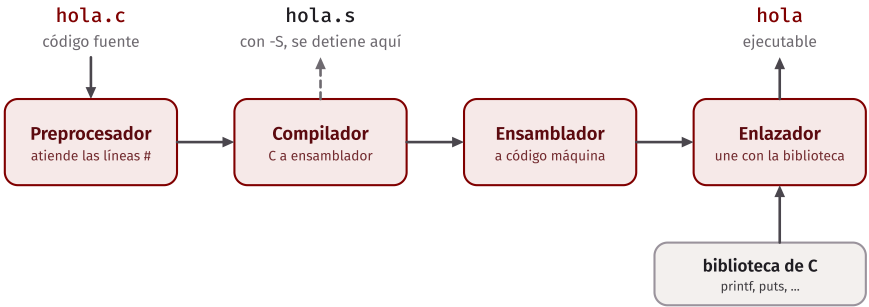
- Una palabra sola se busca en una lista fija de carpetas del sistema. La actual no está.
- `./hola` es una ruta: “el archivo `hola` que está aquí”.

¿Por qué `./hola`?



- Una palabra sola se busca en una lista fija de carpetas del sistema. La actual no está.
- `./hola` es una ruta: “el archivo `hola` que está aquí”.
- `hola.c` es texto, sin permiso de ejecución. Y la CPU no entiende C.

Compilar son cuatro pasos

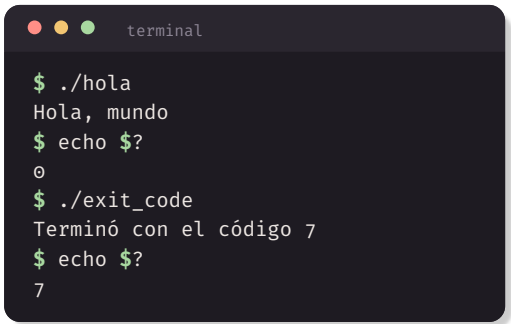


`gcc` hace los cuatro con un solo comando.

Lo que sale de esa cadena

- El código de `printf` no está en `stdio.h`: está en la biblioteca de C, que se carga al correr el programa.
- El ejecutable sirve para una familia de procesadores y un sistema operativo.
- El código fuente se puede volver a compilar en otra máquina.

`return` le responde al shell



- `echo $?` escribe el resultado del último comando.
- Por convención, 0 significa que todo salió bien.
- Cualquier otro valor indica algún problema.

Un programa que no es un algoritmo

change_780.c

```c
#include <stdio.h>

int main(void) {
    printf("Vuelto de S/ 7.80:\n");
    printf("S/ 5.00\n");
    printf("S/ 2.00\n");
    printf("S/ 0.50\n");
    printf("S/ 0.20\n");
    printf("S/ 0.10\n");
    printf("Total: 5 monedas\n");
    return 0;
}
```

- Solo recita una respuesta calculada a mano.

Un programa que no es un algoritmo

change_780.c

```c
#include <stdio.h>

int main(void) {
    printf("Vuelto de S/ 7.80:\n");
    printf("S/ 5.00\n");
    printf("S/ 2.00\n");
    printf("S/ 0.50\n");
    printf("S/ 0.20\n");
    printf("S/ 0.10\n");
    printf("Total: 5 monedas\n");
    return 0;
}
```

- Solo recita una respuesta calculada a mano.
- Con S/ 3.40, habría que reescribirlo.

Un programa que no es un algoritmo

change_780.c

```c
#include <stdio.h>

int main(void) {
    printf("Vuelto de S/ 7.80:\n");
    printf("S/ 5.00\n");
    printf("S/ 2.00\n");
    printf("S/ 0.50\n");
    printf("S/ 0.20\n");
    printf("S/ 0.10\n");
    printf("Total: 5 monedas\n");
    return 0;
}
```

- Solo recita una respuesta calculada a mano.
- Con S/ 3.40, habría que reescribirlo.
- Para calcular necesita guardar valores y repetir: semanas 2 y 5.

SECCIÓN 08

Cuando algo sale mal

08

Dos tipos de error

Tipo	Cuándo aparece	¿Hay ejecutable?
Error de compilación	Al correr `gcc`	No
Error de ejecución	Al correr el programa	Sí, se compiló bien

Advertencia ***(warning)***

No detiene la compilación, pero algo es sospechoso. En este curso el código no debe dar advertencias.

Sin el punto y coma

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:28: error: expected ‘;’ before ‘return’
    4 |     printf("Hola, mundo\n")
      |                            ^
      |                            ;
    5 |     return 0;
      |     ~~~~~~
```

Sin el punto y coma

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:28: error: expected ‘;’ before ‘return’
    4 |     printf("Hola, mundo\n")
      |                            ^
      |                            ;
    5 |     return 0;
      |     ~~~~~~
```

- `hola.c:4:28`: archivo, línea, columna.

Sin el punto y coma

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:28: error: expected ‘;’ before ‘return’
    4 |     printf("Hola, mundo\n")
      |                            ^
      |                            ;
    5 |     return 0;
      |     ~~~~~~
```

- `hola.c:4:28`: archivo, línea, columna.
- “Se esperaba `;` antes de `return`”.

Sin el punto y coma

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:28: error: expected ‘;’ before ‘return’
    4 |     printf("Hola, mundo\n")
      |                            ^
      |                            ;
    5 |     return 0;
      |     ~~~~~~
```

- `hola.c:4:28`: archivo, línea, columna.
- “Se esperaba `;` antes de `return`”.
- Nombra `return`, de la línea 5, pero el `;` falta al final de la línea 4.

Sin la comilla: tres errores

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:12: warning: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^
hola.c:4:12: error: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^~~~~~~~~~~~~~~
hola.c:5:5: error: expected expression before ‘return’
    5 |     return 0;
      |     ^~~~~~
hola.c:5:14: error: expected ‘;’ before ‘}’ token
    5 |     return 0;
      |              ^
      |              ;
    6 | }
      | ~
```

- La línea 5 está bien.

Sin la comilla: tres errores

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:12: warning: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^
hola.c:4:12: error: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^~~~~~~~~~~~~~~
hola.c:5:5: error: expected expression before ‘return’
    5 |     return 0;
      |     ^~~~~~
hola.c:5:14: error: expected ‘;’ before ‘}’ token
    5 |     return 0;
      |              ^
      |              ;
    6 | }
      | ~
```

- La línea 5 está bien.
- `gcc` pierde el hilo después del primer error.

Sin la comilla: tres errores

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:12: warning: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^
hola.c:4:12: error: missing terminating " character
    4 |     printf("Hola, mundo\n);
      |            ^~~~~~~~~~~~~~~
hola.c:5:5: error: expected expression before ‘return’
    5 |     return 0;
      |     ^~~~~~
hola.c:5:14: error: expected ‘;’ before ‘}’ token
    5 |     return 0;
      |              ^
      |              ;
    6 | }
      | ~
```

- La línea 5 está bien.
- `gcc` pierde el hilo después del primer error.

Regla

Lean el **primer** error, corríjanlo y vuelvan a compilar.

El mismo programa de antes

terminal

```
$ gcc -Wall -Wextra -std=c17 hola.c -o hola
hola.c: In function ‘main’:
hola.c:4:28: error: expected ‘;’ before ‘return’
    4 |     printf("Hola, mundo\n")
      |                            ^
      |                            ;
    5 |     return 0;
      |     ~~~~~~
$ ls
hola  hola.c
$ ./hola
Hola, mundo
```

Error común

Si `gcc` halla un error en el código, el ejecutable anterior sigue ahí. `./hola` corre el programa **viejo**.

Lean lo que dice `gcc` antes de ejecutar.

Errores de ejecución: sin ningún aviso

terminal

```
$ gcc -Wall -Wextra -std=c17 wrong_change.c -o wrong_change
$ ./wrong_change
Vuelto de S/ 7.80:
S/ 5.00
S/ 2.00
S/ 1.00
S/ 0.50
S/ 0.20
S/ 0.10
Total: 6 monedas
```

- `gcc` no dice nada. El programa termina normalmente.

Errores de ejecución: sin ningún aviso

terminal

```
$ gcc -Wall -Wextra -std=c17 wrong_change.c -o wrong_change
$ ./wrong_change
Vuelto de S/ 7.80:
S/ 5.00
S/ 2.00
S/ 1.00
S/ 0.50
S/ 0.20
S/ 0.10
Total: 6 monedas
```

- `gcc` no dice nada. El programa termina normalmente.
- Esas monedas suman S/ 8.80: un sol de más.

Errores de ejecución: sin ningún aviso

terminal

```
$ gcc -Wall -Wextra -std=c17 wrong_change.c -o wrong_change
$ ./wrong_change
Vuelto de S/ 7.80:
S/ 5.00
S/ 2.00
S/ 1.00
S/ 0.50
S/ 0.20
S/ 0.10
Total: 6 monedas
```

- `gcc` no dice nada. El programa termina normalmente.
- Esas monedas suman S/ 8.80: un sol de más.
- Solo se nota al comparar con lo que debía salir.

Errores de ejecución: dividir entre cero

terminal

```
$ gcc -Wall -Wextra -std=c17 divide_by_zero.c -o divide_by_zero
divide_by_zero.c: In function ‘main’:
divide_by_zero.c:6:23: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("%d\n", 10 / 0);
      |                       ^
$ ./divide_by_zero
Antes de dividir
0
Después de dividir
```

- `%d`: aquí va un entero. Lo verán la próxima semana.

Errores de ejecución: dividir entre cero

terminal

```
$ gcc -Wall -Wextra -std=c17 divide_by_zero.c -o divide_by_zero
divide_by_zero.c: In function ‘main’:
divide_by_zero.c:6:23: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("%d\n", 10 / 0);
      |                       ^
$ ./divide_by_zero
Antes de dividir
0
Después de dividir
```

- `%d`: aquí va un entero. Lo verán la próxima semana.
- Ese 0 no significa nada. C no define qué pasa.

Errores de ejecución: dividir entre cero

terminal

```
$ gcc -Wall -Wextra -std=c17 divide_by_zero.c -o divide_by_zero
divide_by_zero.c: In function ‘main’:
divide_by_zero.c:6:23: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("%d\n", 10 / 0);
      |                       ^
$ ./divide_by_zero
Antes de dividir
0
Después de dividir
```

- `%d`: aquí va un entero. Lo verán la próxima semana.
- Ese 0 no significa nada. C no define qué pasa.
- Depende de la máquina: en otro procesador puede cortarse.

Controles mientras corre el programa

terminal

```
$ gcc -Wall -Wextra -std=c17 -g -fsanitize=address,undefined divide_by_zero.c -o divide_by_zero
divide_by_zero.c: In function ‘main’:
divide_by_zero.c:6:23: warning: division by zero [-Wdiv-by-zero]
    6 |     printf("%d\n", 10 / 0);
      |                     ^
$ ./divide_by_zero
Antes de dividir
divide_by_zero.c:6:23: runtime error: division by zero
0
Después de dividir
```

En los laboratorios se compila además con `-g -fsanitize=address,undefined`. Avisa al correr, con archivo, línea y columna.

SECCIÓN 09

El laboratorio

09

Esta semana en el laboratorio

1. Abrir el entorno y correr `env/check.sh`: todas las líneas dicen `ok`.
2. Moverse con la terminal y crear la carpeta `lab01` dentro de `code/week-01/lab`.
3. Escribir `hola.c`, compilarlo y ejecutarlo.
4. Borrar a propósito un `;` y una comilla. Anotar en `compiler_errors.txt` lo que dice `gcc`.
5. Escribir el vuelto de S/ 7.80 en pseudocódigo y seguirlo a mano en `change.txt`.

Comprobar

Desde `code/week-01`, `make test-lab` revisa su trabajo. Falla hasta que lo terminen.

Para la próxima clase

- Leer los apuntes de la semana 1 y hacer los ejercicios.
- Terminar el laboratorio.
- La semana 2: variables, tipos, operadores, y leer datos del usuario.

Pregunta

Escriban un algoritmo que reciba tres números y escriba el mayor. Síganlo a mano con 4, 9, 2.