

# TEMA 1

## Introducción: Conceptos Básicos

Emma Rollón  
erollon@cs.upc.edu

Departamento de Ciencias de la Computación

- ❶ Algoritmo, Lenguaje de programación, Programa informático
- ❷ Etapas en la construcción de un programa
- ❸ Paradigma imperativo
  - datos
    - tipos de datos
    - variables
    - expresiones
  - instrucciones
    - declarar y asignar
    - entrada / salida
    - condicional
    - iterativa
- ❹ Visibilidad de variables
- ❺ Ejercicios
  - Suma de dígitos ( $\simeq$  número de dígitos)
  - Producto  $X * Y$  (versión rápida); Potencia  $X^Y$
  - Número de a's en secuencia de caracteres
  - Dibujar un triángulo

# Algoritmo, Lenguaje de programación, Programa informático

## **Algoritmo:**

Descripción precisa de cómo pasar de una situación inicial a una situación final. Es el método de resolución de un problema.

## **Lenguaje de programación:**

Lenguaje formal capaz de expresar de forma precisa algoritmos que pueden ser llevados a cabo por máquinas como las computadoras.

## **Programa informático:**

Algoritmo escrito en un lenguaje de programación que realiza una cierta tarea.

## **Observación**

El algoritmo es independiente del lenguaje de programación.

# Algoritmo, Lenguaje de programación, Programa informático



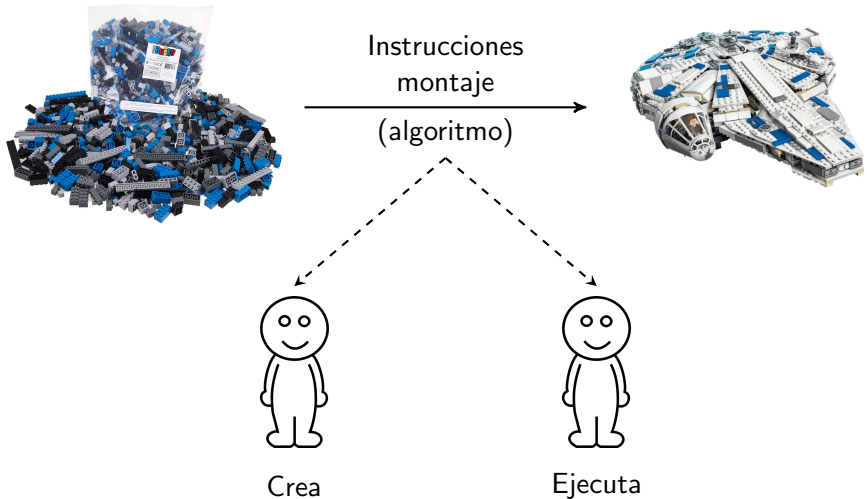
Instrucciones  
montaje  
→  
(algoritmo)



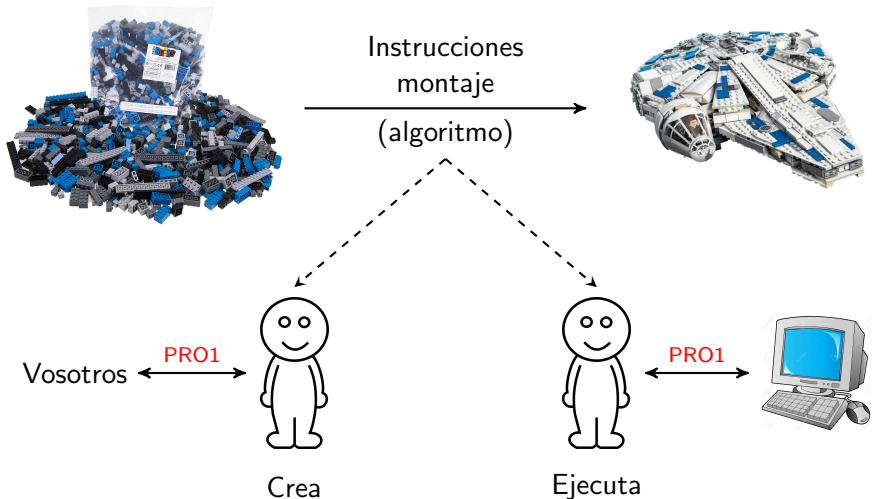
Pasos  
→  
(algoritmo)



# Algoritmo, Lenguaje de programación, Programa informático



# Algoritmo, Lenguaje de programación, Programa informático



# Etapas en la construcción de un programa informático

**Especificación:** describe lo que tiene que hacer nuestro programa (**QUÉ**):

- Precondición: describe la situación inicial. Condiciones de los datos de los que parto ('reglas del juego').
- Postcondición: describe la situación final. Condiciones de los datos a los que llego (lo que quiero conseguir).





**Diseño:** algoritmo (método) para resolver el problema (**CÓMO**).



**Implementación:** escribir el algoritmo con un lenguaje de programación determinado.

**Ejecución y pruebas:** ejecutar el programa probando con diferentes ejemplos de situaciones iniciales para verificar que los datos en la situación final son los esperados. Cada par (datos entrada, datos salida) se denomina juego de pruebas.

# Etapas: ejemplo

**Statement**





Feu un programa que, donada una quantitat de segons, digui quantes hores, minuts i segons representa.



**Entrada**

L'entrada consisteix en un natural  $n$ .

**Sortida**

Escriuiu tres naturals  $h, m, s$  tals que  $3600h+60m+s=n$ , amb  $m<60$  i  $s<60$ .

**Public test cases**



Input	Output
3661	1 1 1
0	0 0 0
76234	21 10 34



## Especificación:

- Lenguaje natural:
  - Precondición: dado un número natural  $n$  que representa segundos
  - Postcondición: escribe la descomposición en horas  $h$ , minutos  $m$ , y segundos  $s$  de  $n$ .
- Lenguaje matemático:
  - Precondición:  $n \in \mathbb{N}$
  - Postcondición:  $3600h + 60m + s = n, 0 \leq s, m < 60, h, m, s \in \mathbb{N}$

# Etapas: Ejemplo

## Diseño:

### Algoritmo 1:

- leer el número  $n$
- $h$  es  $n / 3600$
- $m$  es  $(n \text{ módulo } 3600) / 60$
- $s$  es  $n \text{ módulo } 60$
- escribir  $h, m, s$

### Algoritmo 2:

- leer el número  $n$
- $s$  es  $n \text{ módulo } 60$
- $m$  es  $(n / 60) \text{ módulo } 60$
- $h$  es  $n / 3600$
- escribir  $h, m, s$

## Observación

El algoritmo (método) para resolver un problema no es necesariamente único. Cada algoritmo tendrá un coste en cuanto a tiempo y espacio. En cada situación, tendremos que escoger el mejor, aunque no siempre la respuesta es obvia.

# Etapas: Ejemplo

## Implementación algoritmo 1:

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int h = n/3600;
    int m = (n%3600)/60;
    int s = n%60;
    cout << h << " " << m << " " << s << endl;
}
```

# Etapas: Ejemplo

## Ejecución y pruebas:

- Juego de pruebas 1:      E: 0              S: 0 0 0
- Juego de pruebas 2:      E: 3600          S: 1 0 0
- Juego de pruebas 3:      E: 60              S: 0 1 0
- ...
- Juego de pruebas Y:      E: -14            S: ?? **La entrada NO es válida**

## Observación

El programa sólo tiene que funcionar para aquellos datos de entrada que cumplen la precondición!!

# Paradigma Imperativo

El programa se describe en función de un conjunto de variables que almacenan valores (**DATOS**) y de un conjunto de sentencias que se ejecutan secuencialmente y manipulan esas variables (**INSTRUCCIONES**).

## Sobre los datos:

Tipos de datos  $\equiv$  conjunto de valores + operaciones

Variables  $\equiv$  nombre + tipo de datos

Expresiones  $\equiv$  variables + literales + operadores + funciones

## Sobre las instrucciones:

Asignación

Entrada / Salida

Condicional

Iterativa

# Tipos de datos

Tipo de datos  $\equiv$  conjunto de valores + operaciones

Tipo de datos	Valores	Operaciones	Operaciones lógicas
<b>int</b>	$\pm(2^{31} - 1)$	$+, -, *, /, \%$	$==, !=,$ $>, <,$ $>=, <=$
<b>double</b>	$\pm 1,7976 \times 10^{308}$	$+, -, *, /$	
<b>bool</b>	true, false	not, and, or	
<b>char</b>	'a', 'b', ..., 'A', 'B', ..., '0', ..., '9', '.', ...	$+, -$	
<b>string</b>	"Buenos días!", "sala", ..., "A"		

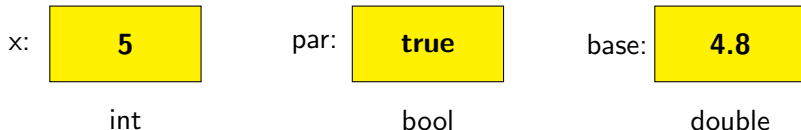
Cuidado con el operador / que representa:

- división entera (cuando los dos operandos son enteros)
- división real (cuando al menos uno de los dos operandos es real)

# Variables

Variable  $\equiv$  nombre + tipo de datos

Nombre simbólico para representar valores. Puedo pensar que es una caja que puede contener un valor de un tipo de datos determinado.



Las variables se pueden

- Crear (declarar)
- Utilizar
  - Consultar su valor
  - Actualizar su valor

Expresión  $\equiv$  variables + literales + operadores + funciones

Combinación de variables, literales, operadores y funciones sintácticamente correcta (es decir, se puede evaluar y retorna un valor).

La expresión es del tipo de datos al que evalúa. Por ejemplo, es una expresión entera si el valor al que evalúa es un entero, es una expresión real si el valor al que evalúa es un real, ...

## Ejemplos:

- $5 == 6$  (evaluable  $\rightarrow$  false)
- $5 * 6 + 4$  (evaluable  $\rightarrow$  34)
- $6/4*$  (NO evaluable: el operador  $*$  es binario)
- $6/4*\text{true}$  (NO evaluable: los tipos no son compatibles)



## Reglas de precedencia de los operadores (de más a menos prioritario):

Unary	+, -, not
Multiplicative	* / %
Additive	+ -
Relational (inequalities)	> >= < <=
Relational (equalities)	== !=
Conjunction	and
Disjunction	or

Para cambiar la precedencia se utilizan los paréntesis. Por ejemplo,  $5 * (6 + 4)$ .

## Expresiones con booleanos:

x	y	x and y	x or y
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

x	not x
False	True
True	False

# Instrucciones básicas

## Plantilla de programación

```
#include <iostream>
using namespace std;

int main() {
    ... ;    // conjunto de instrucciones
    ... ;    // que se ejecutan secuencialmente
    ... ;    // y en orden (de primera a última)
}
```

- Las instrucciones que veremos a partir de ahora van dentro del bloque main.
- Los caracteres // indican que a partir de ahí y hasta el final de la línea es un comentario (para el ordenador es como si no estuvieran).

# Instrucciones básicas: Declaración de variable

## Sintaxis:

```
tipo_de_datos nombre_variable;
```

## Semántica:

- A partir de esa instrucción, la variable `nombre_variable` existe (el programa puede utilizarla).
- Después de declarar una variable, su valor es **inválido!!**
- Sólo podrá contener valores del tipo `tipo_de_datos`.

## Ejemplo:

```
int x;  
string s;  
int a, b, c; // declaramos más de una var a la vez
```

# Instrucciones básicas: Asignación

## Sintaxis:

```
nombre_variable = expresión;
```

## Semántica:

- La variable toma el valor al que evalúa la expresión.
- El valor previo de la variable (si tenía alguno) se pierde.
- Expresión y variable han de ser del mismo (compatible) tipo de datos.

## Ejemplo 1:

```
int x, y;  
x = 5;  
y = 10;  
x = 5*x + y;  
y = x - 2;
```

## Ejemplo 2: (intercambio valor)

```
int x, y;  
x = 5;  
y = 10;  
int aux = x;  
x = y;  
y = aux;
```

# Instrucciones básicas: Entrada/Salida

## Sintaxis:

```
cin >> nombre_variable;    // entrada
cout << expresión;         // salida
```

## Semántica:

- cin: se asigna a la variable lo que el usuario escriba por teclado.
- cout: escribe por pantalla el resultado de evaluar la expresión.

## Ejemplo:

```
int main() {
    int x, y;
    cin >> x >> y;
    cout << "El doble de " << x << " es: " << 2*x << endl;
    cout << "La mitad de " << y << " es: " << y/2 << endl;
}
```

# ¿Qué más necesitamos?

## Statement



Escribe un programa que lea dos números y escriba su mínimo.

### Entrada

La entrada está formada por dos enteros.

### Salida

Escribid una línea con el mínimo de los dos números.

## Public test cases



### Input

634 371

### Output

371

### Input

-31 -21

### Output

-31

### Input

23 23

### Output

23

# Instrucciones básicas: Condicional (v1)

## Sintaxis:

```
if (E) {  
    I1;  
    ...;  
    In;  
}
```

## Semántica

- Si la expresión booleana E evalúa a cierto, entonces se ejecutan las instrucciones I1, I2, ..., In. Si E evalúa a falso, no se ejecutan.
- Una vez hecho, se ejecuta la instrucción que sigue al condicional.

## Ejemplo 1 (valor absoluto)

```
int x;  
cin >> x;  
if (x < 0) x = -x;  
cout << x << endl;
```

## Ejemplo 2 (valor absoluto - mala)

```
int x;  
cin >> x;  
if (x < 0) cout << -x << endl;  
if (x >= 0) cout << x << endl;
```

# Instrucciones básicas: Condicional (v2)

## Sintaxis:

```
if (E) {  
    I1;  
    ...;  
    In;  
} else {  
    I'1;  
    ...;  
    I'm;  
}
```

## Ejemplo 1 (valor absoluto)

```
int x;  
cin >> x;  
if (x < 0) cout << -x << endl;  
else cout << x << endl;
```

## Semántica:

- Si la expresión booleana E es cierta, se ejecutan las instrucciones I1, ..., In.
- Si no (es decir, si E evalúa a falso), entonces se ejecutan las instrucciones I'1, ..., I'm.
- Una vez hecho, se ejecuta la instrucción que sigue al condicional.

## Ejemplo 2 (mínimo de 2 enteros)

```
int x, y;  
cin >> x >> y;  
if (x < y) cout << x << endl;  
else cout << y << endl;
```



# Instrucciones básicas: Condicional (v3)

## Sintaxis:

```
if (E1) {  
    ...;  
} else if (E2) {  
    ...;  
} else if (E3) {  
    ...;  
} else {  
    ...;  
}
```

## Semántica:

- Se evalúan las expresiones booleanas E1, ..., E3 en orden hasta encontrar una que sea cierta y se ejecutan las instrucciones asociadas a ese bloque.
- Si no hay ninguna que sea cierta, entonces se ejecutan las instrucciones asociadas al bloque else.
- Una vez hecho, se ejecuta la instrucción que sigue al condicional.

# Instrucciones básicas: Condicional (v3)

## Ejemplo (máximo de tres enteros)

### Especificación:

- Pre: dados tres enteros
- Post: escribe el máximo de ellos.

### Diseño:

- 1 Leer 3 enteros que llamaré  $x$ ,  $y$ ,  $z$ .
- 2 Comprobar si  $x$  es el mayor de los tres. Si lo es, escribirlo y acabar.
- 3 Si  $x$  no es el mayor, comprobar si  $y$  es mayor que  $z$ . Si lo es, escribirlo y acabar.
- 4 Si ni  $x$  ni  $y$  son el máximo, entonces escribir  $z$  (porque es el mayor) y acabar.

# Instrucciones básicas: Condicional (v3)

## Ejemplo (máximo de tres enteros)

### Implementación:

```
int main() {  
    int x, y, z;  
    cin >> x >> y >> z;  
  
    if (x >= y and x >= z) cout << x << endl;  
    else if (y >= z) cout << y << endl;  
    else cout << z << endl;  
}
```

## Observaciones

- Sólo se ejecutan las instrucciones del primer bloque cuya condición es cierta.
- Que las condiciones sean falsas también nos aporta conocimiento.

# Instrucciones básicas: Condicional (v3)

## Ejemplo (máximo de tres enteros)

### Implementación:

```
int main() {  
    int x, y, z;  
    cin >> x >> y >> z;  
  
    if (x >= y and x >= z) cout << x << endl;  
    else if (y >= z) cout << y << endl;  
    else cout << z << endl;  
}
```

### Juegos de pruebas:

- El máximo es el primer número: 8 4 3
- El máximo es el segundo número: 4 8 3
- El máximo es el tercer número: 4 3 8
- Todos los números son iguales: 8 8 8

# ¿Qué más necesitamos?

## Statement



Feu un programa que llegeixi un nombre  $n$ , i que escrigui tots els nombres entre 0 i  $n$ .

### Entrada

L'entrada consisteix en un natural  $n$ .

### Sortida

Escriviu en ordre tots els naturals entre 0 i  $n$ .

## Public test cases



### Input

5

### Output

0  
1  
2  
3  
4  
5

# Instrucciones básicas: Bucle (while)

## Sintaxis:

```
while (E) {  
    I1;  
    ...;  
    In;  
}
```

## Semántica:

- Si la expresión booleana E es cierta, entonces se ejecutan las instrucciones I1, ..., In.
- Si la expresión booleana E es cierta, entonces se ejecutan las instrucciones I1, ..., In.
- ...
- Hasta que la expresión E evalúe a falso. Entonces se sigue por la siguiente instrucción de después del bucle.

# Instrucciones básicas: Bucle (while)

## Sintaxis:

```
while (E) {  
    I1;  
    ...;  
    In;  
}
```

## Nomenclatura:

- E: condición del bucle.
- I1, ..., In: cuerpo del bucle.
- Iteración: cada vez que se ejecuta el cuerpo del bucle.

## Observaciones

- A priori, no sabemos el número de iteraciones que tendrá el bucle.
- Si a mitad de ejecución del cuerpo del bucle E se hace falso, no dejo de ejecutar esas instrucciones.

# Instrucciones básicas: Bucle (while)

## Ejemplo 1:

```
int main() {  
    int x;  
    cin >> x;  
    while (x != 0) {  
        cout << x << endl;  
        x = x - 1;  
    }  
}
```

## Cuestiones:

- Qué hace el código si le doy un 6 en la entrada?
- Qué escribe en la última iteración?
- Si cambio el orden de las instrucciones del cuerpo del bucle, qué hace con entrada 6?
- Qué pasa si la entrada es -5?



# Instrucciones básicas: Bucle (while)

## Ejemplo 2:

### Especificación:

- Pre: dado un número natural  $n$  ( $n \geq 0$ )
- Post: escribe los números  $0\ 1\ 2\ \dots\ n$ , cada uno en una línea.

### Diseño:

- 1 Leer número  $n$ .
- 2 Empezando desde el valor  $0$  ...
- 3 ... comprobaré si ese valor es menor o igual que  $n$ . Si lo es,
  - escribiré ese valor, y lo incrementaré en  $1$ .
- 4 Volveré al punto 3 hasta que el valor sea mayor que  $n$ .

# Instrucciones básicas: Bucle (while)

## Ejemplo 2:

### Diseño:

- ❶ Leer número  $n$
- ❷ Empezando desde el valor 0 ...
- ❸ ... comprobaré si ese valor es menor o igual que  $n$ . Si lo es,
  - escribiré ese valor, y lo incrementaré en 1.
- ❹ Volveré al punto 3 hasta que el valor sea mayor que  $n$ .

### Implementación:

```
int n;  
cin >> n;  
int i = 0;  
while (i <= n) {  
    cout << i << endl;  
    i = i + 1;  
}
```

# Instrucciones básicas: Bucle (for)

## Sintaxis:

```
for (Ini; E; Modif) {  
    I1;  
    ...;  
    In;  
}
```

**Ini** inicialización de la variable del bucle

**E** expresión booleana sobre la variable del bucle

**Modif** modificación de la variable del bucle

## Semántica:

- Ejecuto Ini
- si E es cierta, entonces ejecuto I1; ..., In; Modif;
- si E es cierta, entonces ejecuto I1; ..., In; Modif;
- ...
- hasta que E sea falso.

# Instrucciones básicas: Bucle (for)

## Sintaxis:

```
for (Ini; E; Modif) {  
    I1;  
    ...;  
    In;  
}
```

**Ini** inicialización de la variable del bucle

**E** expresión booleana sobre la variable del bucle

**Modif** modificación de la variable del bucle

## Equivalencia (en cuanto a ejecución):

```
Ini;  
while (E) {  
    I1;  
    ...;  
    In;  
    Modif;  
}
```

### Observación

Hay una diferencia sutil. Lo veremos en **visibilidad de variables**.

# Instrucciones básicas: Bucle (for)

## Ejemplo 1:

```
int main() {  
    for (int i = 0; i < 100; ++i) {  
        cout << i << endl;  
    }  
}
```

## Cuestiones:

- Qué escribe este código?
- Cuántas veces itera el bucle?

## Ejemplo 2:

```
int main() {  
    for (int i = 0; i < 100; ++i) {  
        cout << i << endl;  
        i = 100;    // lo puedo hacer? NO modificar var for!  
    }  
}
```

# Visibilidad de variables

Las variables existen (son visibles) a lo largo de todo el código?  $\implies$  NO

- Existen a partir de que se declaran
- Sólo dentro del bloque en el que se declara

Qué bloques hemos visto hasta ahora?

- Programa principal (main)
- Cuerpo de un bucle
- Ramas de un condicional

Por tanto, un bloque puede contener otros bloques. Cuál es la visibilidad de variables entre bloques?

- Las variables de los bloques contenedores (externos) son visibles (salvo una excepción) en los bloques contenidos (internos).
- Las variables de los bloques contenidos (internos), no son visibles en los bloques contenedores (externos).

# Visibilidad de variables: Ejemplo 1

```
int main() {  
    cout << "Empezamos ejecución" << endl;  
    // En este punto todavía no hay ninguna variable visible  
    int x;  
    // A partir de aquí existe la var x  
    cin >> x;  
    while (x > 0) {  
        // Aquí la var x existe, la var d no existe  
        int d = x%10;  
        // Aquí la var x existe, la var d existe  
        cout << d;  
        x = x/10;  
    }  
    // Aquí la var x existe, la var d no existe  
    cout << endl;  
}
```

# Visibilidad de variables: Ejemplo 2

Qué pasa si en dos bloques anidados existe un mismo nombre de variable?

```
int main() {  
    int x = 10;  
    int y = 20;  
    while (x >= 0) {  
        int y = 3;  
        cout << x*y << endl;    // qué y? la de valor 3  
        x = x - 1;  
    }  
    cout << y << endl;    // qué y? la de valor 20  
}
```

La variable visible será la declarada en el bloque más cercano a donde se usa. La variable del bloque interno enmascara (oculta) la del bloque externo.

## Observación

Para evitar errores, NO enmascarar variables.



# Visibilidad de variables: Ejemplo 3

A) En qué bloques es visible la variable `i` en estos dos códigos?

```
for (int i = 0; i < n; ++i) {  
    cout << i << endl;  
}
```

```
int i = 0;  
while (i < n) {  
    cout << i << endl;  
    ++i;  
}
```

B) Por tanto, qué código es correcto?

```
for (int i = 0; i < n; ++i) {  
    cout << i << endl;  
}  
cout << i << endl;
```

```
int i = 0;  
while (i < n) {  
    cout << i << endl;  
    ++i;  
}  
cout << i << endl;
```

# Ejercicios: suma de dígitos

```
// Pre: n >= 0
// Post: escribe la suma de los dígitos de n con el formato
//       de los ejemplos siguientes:
// E: 5      S: La suma de los digitos de 5 es 5.
// E: 15     S: La suma de los digitos de 15 es 6.
```

```
int main() {
    int n;
    cin >> n;
    cout << "La suma de los digitos de " << n << " es ";
    int sdigitos = 0;
    while (n > 0) {
        sdigitos = sdigitos + n%10;
        n = n/10;
    }
    cout << sdigitos << "." << endl;
}
```

# Ejercicios: número de dígitos

```
// Pre: n >= 0
// Post: escribe el número de dígitos de n con el formato
//       de los ejemplos siguientes:
// E: 5      S: El numero de digitos de 5 es 1.
// E: 15     S: El numero de digitos de 15 es 2.
```

```
int main() {
    int n;
    cin >> n;
    cout << "El numero de digitos de " << n << " es ";
    int ndigitos = 1;
    while (n > 9) {
        ++ndigitos;
        n = n/10;
    }
    cout << ndigitos << "." << endl;
}
```

# Ejercicios: producto

```
// Pre: dados un entero x; un entero positivo y
// Post: escribe su multiplicación.
// Importante: NO se puede utilizar la operación *
```

```
int main() {
    int x, y;
    cin >> x >> y;
    int prod = 0;
    while (y > 0) {
        prod = prod + x;
        --y;
    }
    cout << prod << endl;
}
```

```
int main() {                                     // Implementación con for
    int x, y;
    cin >> x >> y;
    int prod = 0;
    for (int i = 0; i < y; ++i) prod = prod + x;
    cout << prod << endl;
}
```

# Ejercicios: potencia

```
// Pre: dados un entero x; un entero positivo y
// Post: escribe el valor de x elevado a y.
```

```
int main() {
    int x, y;
    cin >> x >> y;
    int exp = 1;
    while (y > 0) {
        exp = exp*x;
        --y;
    }
    cout << exp << endl;
}
```

```
int main() {                                     // Implementación con for
    int x, y;
    cin >> x >> y;
    int exp = 1;
    for (int i = 0; i < y; ++i) exp = exp*x;
    cout << exp << endl;
}
```

# Ejercicios: número de a's de una secuencia

```
// Pre: dado un natural n y una secuencia de n caracteres
// Post: escribir el número de a's en la secuencia
int n;
cin >> n;
int cont = 0;
for (int i = 0; i < n; ++i) {
    char c;
    cin >> c;
    if (c == 'a') ++cont;
}
cout << cont << endl;
```

```
// Pre: dada una secuencia de caracteres acabada en '.'
// Post: escribir el número de a's en la secuencia
int cont = 0;
char c;
cin >> c;
while (c != '.') {
    if (c == 'a') ++cont;
    cin >> c;
}
cout << cont << endl;
```

```
// Pre: dado una secuencia de caracteres
// Post: escribir el número de a's en la secuencia
int cont = 0;
char c;
while (cin >> c) {
    if (c == 'a') ++cont;
}
cout << cont << endl;
```

# Ejercicios: dibujar un triángulo

```
// Pre: dado un natural n
// Post: escribe un triángulo con n líneas como en los
//       siguientes ejemplos:
// E: 4      S: *
//           **
//           ***
//           ****
//
// E: 2      S: *
//           **

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        // escribir línea i-ésima
        for (int j = 0; j < i; ++j) cout << '*';
        cout << endl;
    }
}
```