PROGRAMACIÓN 1

Grado en Ingeniería Informática e I2ADE

Tema 3

Sentencias de control

Dept. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial

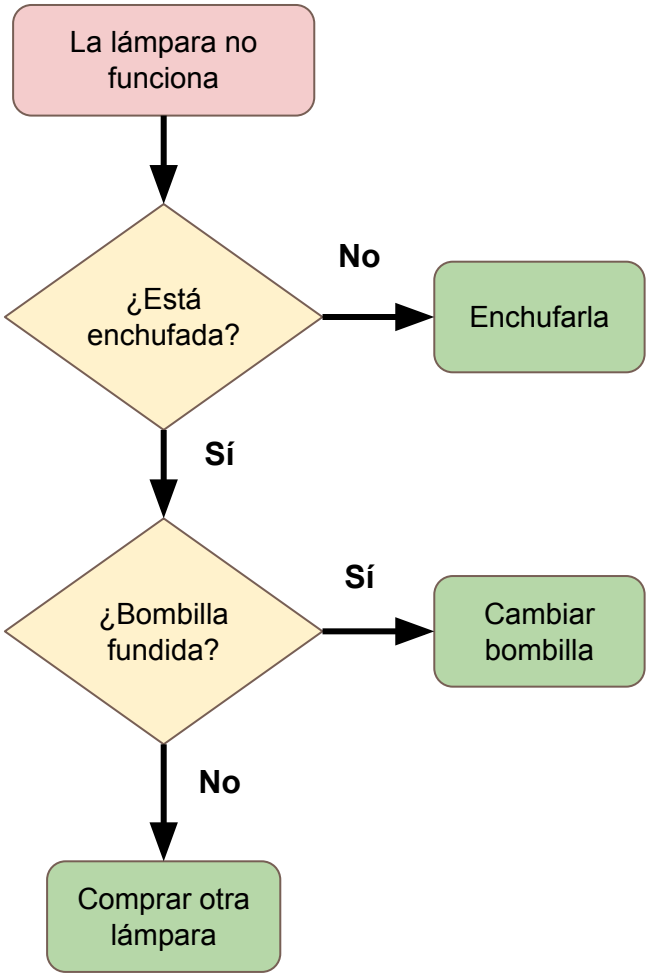
Universitat d'Alacant
Universidad de Alicante

Índice

1. Algoritmos y programas

Concepto de algoritmo

- Un algoritmo es una secuencia ordenada de instrucciones que permiten resolver un problema en un número finito de pasos.
- En informática, los algoritmos son **independientes** tanto **del lenguaje de programación** utilizado como **del ordenador** en el que se ejecutarán.

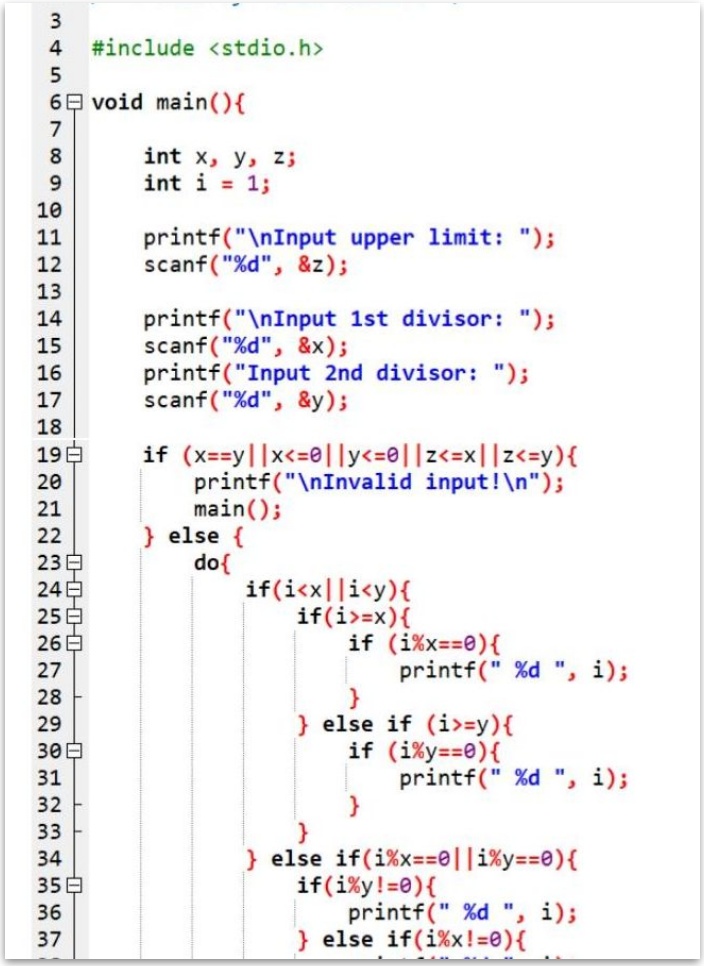


1. Algoritmos y programas

Concepto de programa

- Conjunto de instrucciones (sentencias) ordenadas escritas en un lenguaje de programación para que una computadora lleve a cabo una determinada tarea.

- Un programa informático no es más que un conjunto de algoritmos ordenados y codificados en un lenguaje de programación.

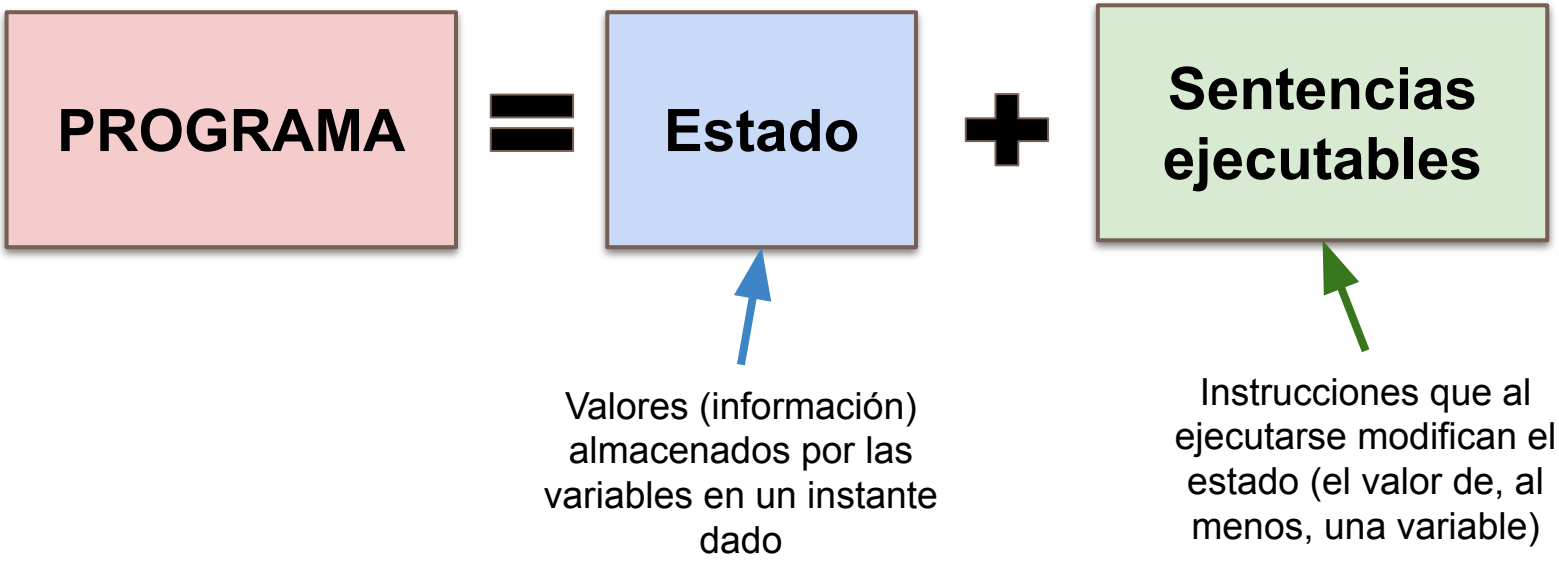


```
3
4   #include <stdio.h>
5
6   void main(){
7
8       int x, y, z;
9       int i = 1;
10
11      printf("\nInput upper limit: ");
12      scanf("%d", &z);
13
14      printf("\nInput 1st divisor: ");
15      scanf("%d", &x);
16      printf("Input 2nd divisor: ");
17      scanf("%d", &y);
18
19      if (x==y||x<=0||y<=0||z<=x||z<=y){
20          printf("\nInvalid input!\n");
21          main();
22      } else {
23          do{
24              if(i<x||i<y){
25                  if(i>=x){
26                      if (i%x==0){
27                          printf(" %d ", i);
28                      }
29                  } else if (i>=y){
30                      if (i%y==0){
31                          printf(" %d ", i);
32                      }
33                  }
34              } else if(i%x==0||i%y==0){
35                  if(i%y!=0){
36                      printf(" %d ", i);
37                  } else if(i%x!=0){
```

1. Algoritmos y programas

Estado de un programa

El estado de un programa en un determinado instante es una configuración única de la información que maneja. Dicho de otra manera, es el valor que tienen cada una de sus variables en ese instante.



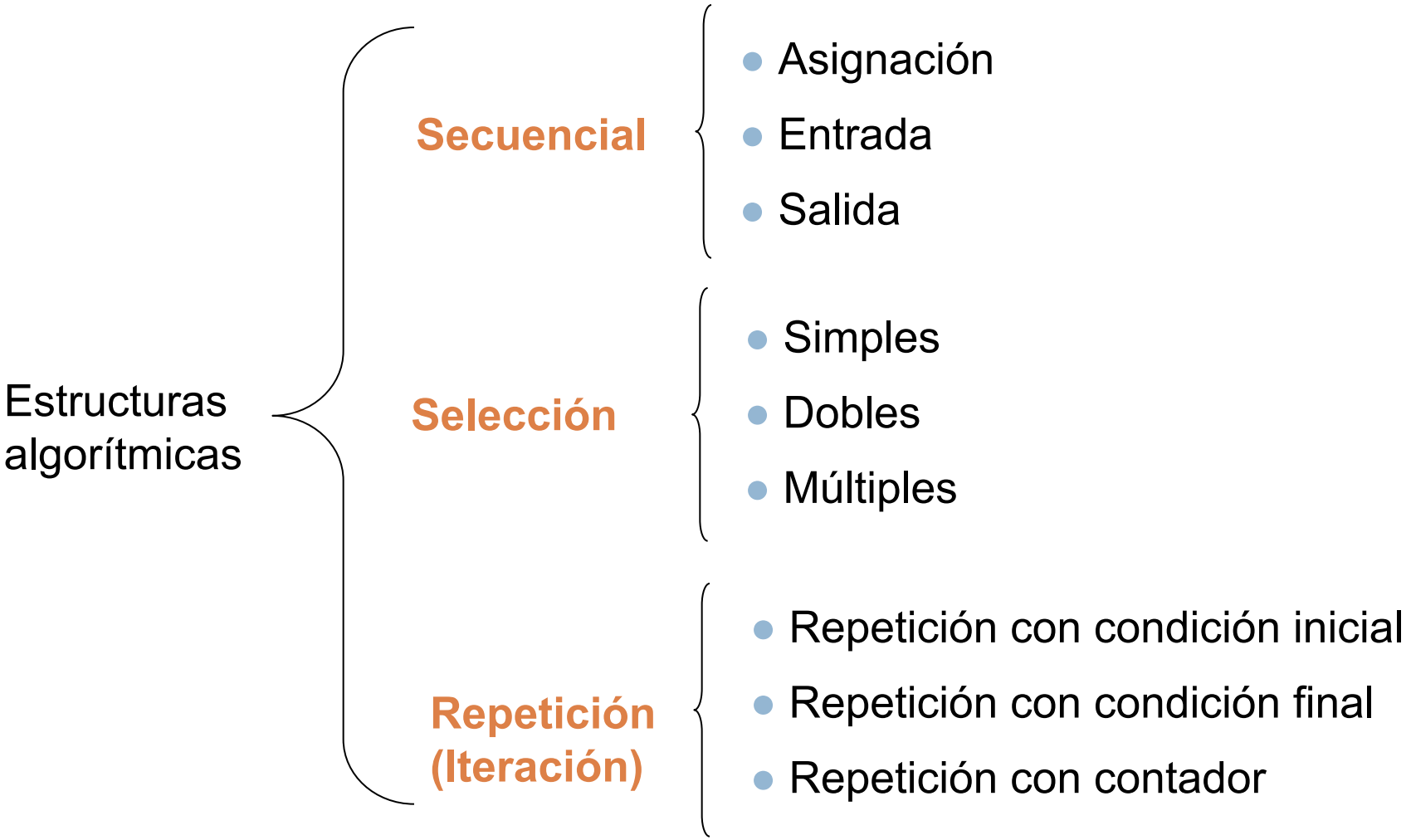
1. Algoritmos y programas

Flujo de control de un algoritmo

- Es el orden en que se ejecutan las instrucciones de un algoritmo.
- Por defecto, el orden en el que se ejecutan las instrucciones es secuencial, de arriba a abajo y de izquierda a derecha.
- Mediante **estructuras algorítmicas** se puede alterar el flujo de control de un algoritmo.

2. Estructuras algorítmicas

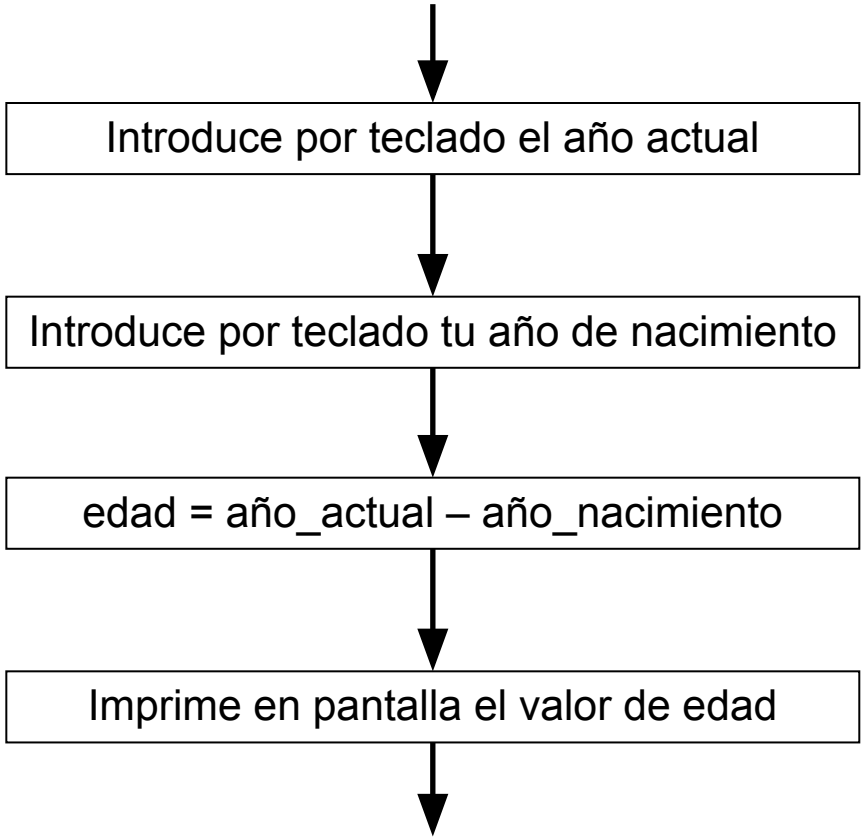
Tipos de estructuras algorítmicas



2. Estructuras algorítmicas

Estructura secuencial

Las acciones (instrucciones) se efectúan una a continuación de otra, de manera consecutiva

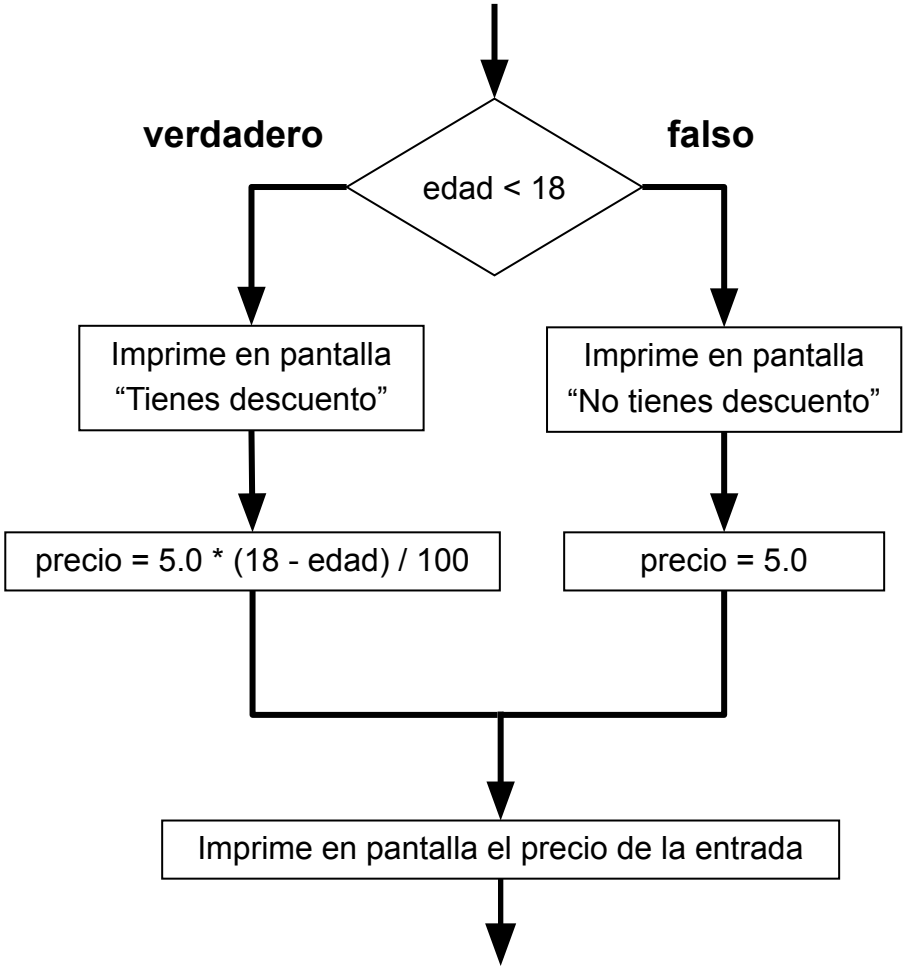


2. Estructuras algorítmicas

Estructura de selección

Permite tomar decisiones entre distintas acciones alternativas dependiendo del valor de una condición.

```
Si condición_cierta Entonces
   <acciones1>
Si_no
   <acciones2>
Fin_Estructura_Selección
```

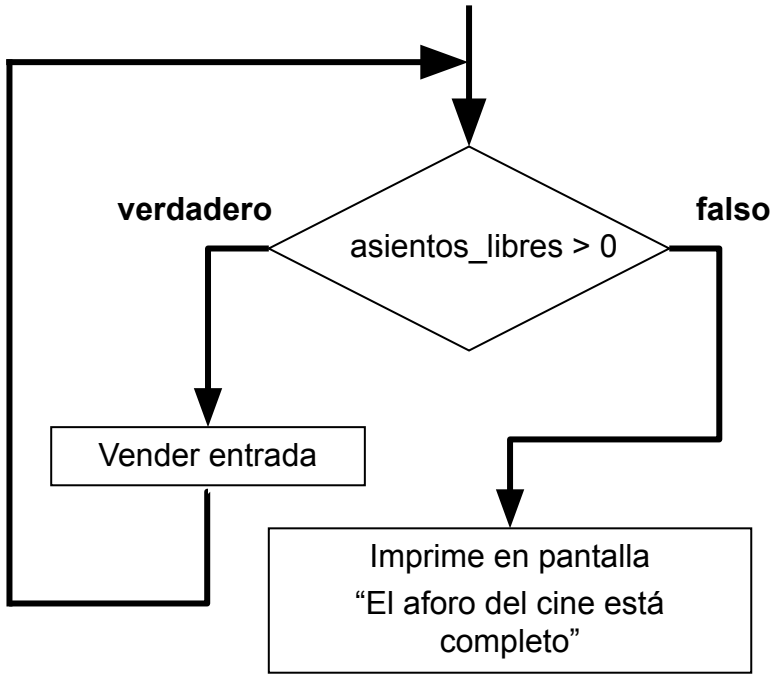


2. Estructuras algorítmicas

Estructura de repetición (iteración)

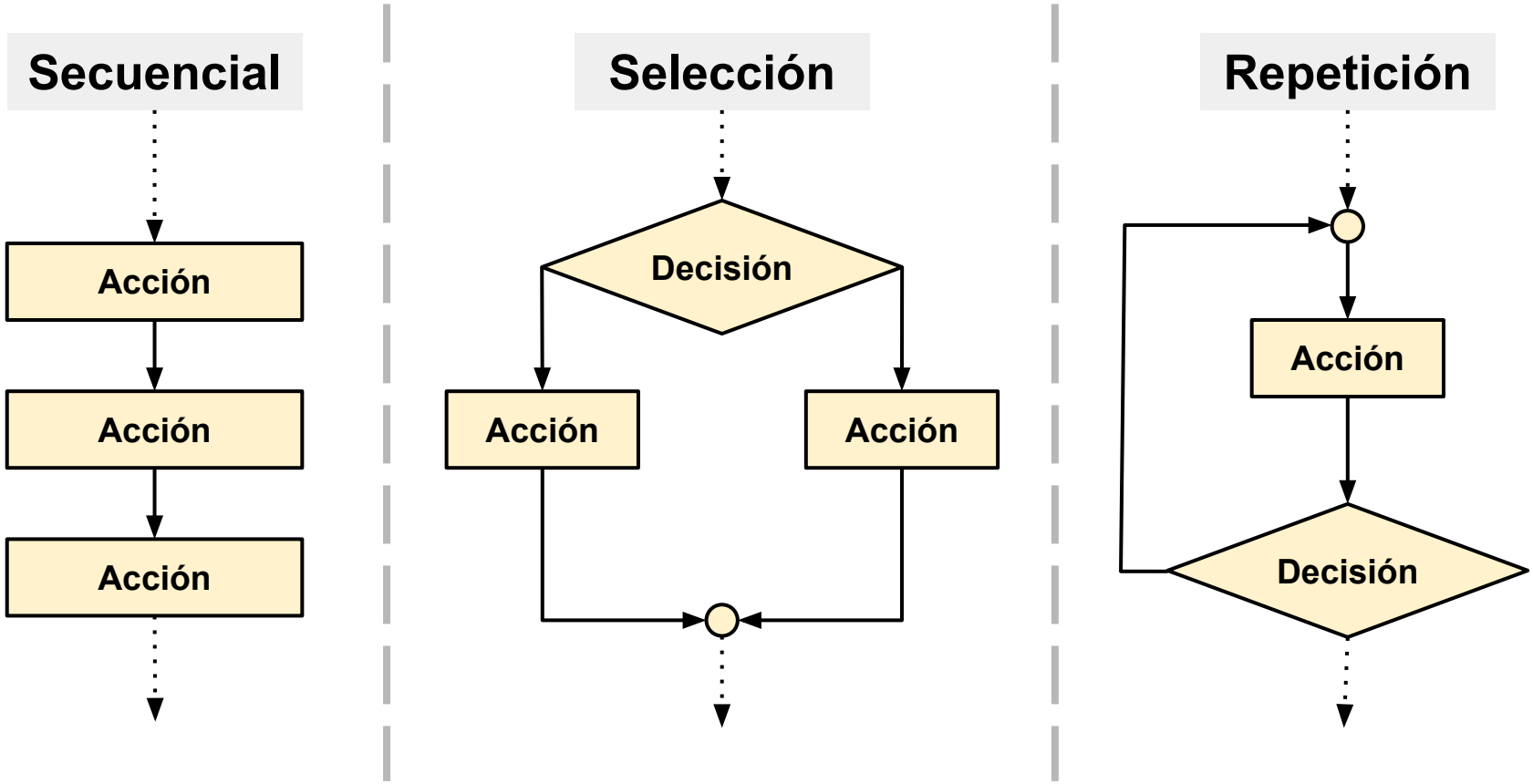
Permite repetir acciones dependiendo del valor de una condición.

```
Mientras condición_cierta Hacer
   <acciones>
Fin_Estructura_Repetición
```



2. Estructuras algorítmicas

Resumen de tipos de estructuras algorítmicas



3. Estructuras de control en C

- Las **estructuras de control o de programación** son las estructuras algorítmicas llevadas a un lenguaje de programación.
- Todos los lenguajes de programación implementan las mismas estructuras de control, aunque puede variar la forma en que se escriben.

3. Estructuras de control de C

Estructuras de control

- **Secuencial**
  - Asignación: Operador `=`
  - Entrada: `scanf`
  - Salida: `printf`
- **Selección**
  - Simples: `if`
  - Dobles: `if-else`
  - Múltiples: `switch` o bien `if-else` ***anidados***
- **Repetición (Iteración)**
  - Repetición con condición inicial: `while`
  - Repetición con condición final: `do-while`
  - Repetición con contador: `for`

Estructuras secuenciales

- **Sentencia de asignación**

```
variable = valor;
```

```
x = 20;
y = 3;
cociente = x / y;
resto = x % y;
```

- **Sentencia de entrada (lectura de datos)**

```
scanf(formato, argumentos);
```

```
scanf(" %d", &x);
scanf(" %f", &y);
```

- **Sentencia de salida (escritura de datos)**

```
printf(formato, argumentos);
```

```
printf("Texto a pantalla");
printf("Valor: %d\n", x);
```

3. Estructuras de control en C

Secuencia de sentencias

- Una secuencia de sentencias puede estar constituida por N sentencias (N ≥ 0)
- Cuando N > 1, la secuencia de sentencias **debe** estar encerrada entre **llaves**

```
{
  secuencia de sentencias
}
```

```
{  // inicio de secuencia de sentencias

  printf("Introduce dos números enteros: ");
  scanf(" %x %y", &x, &y);
  cociente = x / y;
  resto = x % y;
  printf("El cociente es: %f\n", cociente);
  printf("El resto es: %d\n", resto);

}  // fin de la secuencia de sentencias
```

Importante: Todas las sentencias en C terminan con un punto y coma.

Estructuras de selección: sentencia `if`

- Permite decidir si una secuencia de sentencias se van a ejecutar a continuación.

```
if (expresión_lógica) {
   secuencia de sentencias
}
```

```
if (velocidad > 120) {
   printf("AVISO: te pueden multar");
}  // fin de la sentencia if

printf("tu velocidad actual es: %f\n",
        velocidad);
```

- Si el resultado de evaluar *expresión_lógica* es **verdadero**, entonces se ejecuta la secuencia de sentencias asociada a la sentencia **`if`**.
- Si el valor de *expresión_lógica* es **falso**, entonces no se ejecuta la secuencia de sentencias y se pasará a ejecutar la sentencia siguiente al **`if`**.

Importante: Los **paréntesis** que encierran la expresión lógica del **`if`** son **obligatorios**

Estructuras de selección: sentencia `if-else`

- Permite seleccionar entre dos secuencias de sentencias distintas.

```
if (expresión_lógica) {
  secuencia de sentencias 1
}
else {
  secuencia de sentencias 2
}
```

```
if (numero % 2 == 0) {
  printf("El número es par\n");
}
else {
  printf("El número es impar\n");
} // fin de la sentencia if-else

printf("Introduce otro número: ");
```

- Si el valor de *expresión_lógica* es **verdadero** entonces se ejecuta la secuencia de sentencias asociada a la parte del **`if`** (*secuencia de sentencias 1*).
- Si el valor de *expresión_lógica* es **falso** entonces se ejecuta la secuencia de sentencias asociada a la parte del **`else`** (*secuencia de sentencias 2*).

Estructuras de selección: sentencia `if-else` anidada

- Permite seleccionar entre múltiples secuencias de sentencias distintas.

```
if (expresión_lógica_1 ) {
  secuencia de sentencias 1
}
else if (expresión_lógica_2) {
  secuencia de sentencias 2
}
else if (expresión_lógica_3) {
  secuencia de sentencias 3
}
```

```
if (nota >= 9  &&  nota <= 10)
  printf("tu nota es SOBRESALIENTE");
else if (nota >= 7  &&  nota < 9)
  printf("tu nota es NOTABLE");
else if (nota >= 5  &&  nota < 7)
  printf("tu nota es APROBADO");
else if (nota >=0  && nota < 5)
  printf("tu nota es SUSPENSO");
else // última alternativa del if-else anidado
  printf("la nota es incorrecta");
```

- Sólo se ejecuta la secuencia de sentencias asociada a la expresión lógica que primero se evalúe a **verdadero**.
- Si todas las expresiones lógicas se evalúan a **falso**, se ejecutará la secuencia de sentencias del `else`, si existe la alternativa `else`.

Estructuras de selección: sentencia `switch`

- Permite seleccionar entre múltiples secuencias de sentencias distintas. Es equivalente a la estructura `if-else` anidada.

```
switch ( expresión )  {
  case valor_1 : secuencia de sentencias 1;
                 break;
  case valor_2 : secuencia de sentencias 2;
                 break;
  case valor_3 : secuencia de sentencias 3;
                 break;
  default : secuencia de sentencias 4;
}
```

```
switch ( operador ) {
   case '+' :  res = x + y;
               break;
   case '-' :  res = x - y;
               break;
   case '*' :  res = x * y;
               break;
   case '/' :  res = x / y;
               break;
}  // fin de la sentencia switch
printf("Resultado: %f\n", res);
```

- Sólo se ejecuta la secuencia de sentencias asociada al `case` cuyo valor se corresponda con el resultado de la evaluación de *expresión* del `switch`. La sentencia **`break`** determina el fin de la secuencia de sentencias asociada a cada `case`.
- Si el resultado de la expresión del `switch` no se corresponde con ningún valor de un `case`, se ejecuta la secuencia de sentencias asociada a la parte `default` (que es optativa).

Ejercicios

1. Escribe un programa que solicite un número entero al usuario y escriba un mensaje indicando si es par o impar.
2. Escribe un programa que, para un mes (1-12) introducido por el usuario, indique el número de días que tiene (considera un año no bisiesto).
3. Escribe un programa que lea las coordenadas (x,y) de tres puntos de un plano e indique si esos puntos forman un triángulo equilátero.
4. Escribe un programa que visualice tres opciones de un menú y permita al usuario seleccionar una de ellas. Una vez seleccionada la opción, deberá aparecer un mensaje en la pantalla que muestre la opción seleccionada o bien un mensaje de error si la opción es incorrecta:

Ejemplo 1 de ejecución

```
1. Opción1 del menú
2. Opción2 del menú
3. Opción3 del menú

Introduce una opción del menú:  2
La opción seleccionada es 2
```

Ejemplo 2 de ejecución

```
1. Opción1 del menú
2. Opción2 del menú
3. Opción3 del menú

Introduce una opción del menú:  4
La opción seleccionada es incorrecta
```

Estructuras de repetición: bucles

- Un **bucle** es una estructura de programación formada por una secuencia de sentencias, denominada **cuerpo** del bucle, que se puede repetir varias veces.
- Cada ejecución del cuerpo del bucle es una **iteración**.
- El número de veces que se ejecuta el cuerpo del bucle está controlado por una **condición** (expresión lógica).
- Por lo tanto, a la hora de diseñar e implementar un bucle, hay que tener en cuenta dos aspectos:
    1. **¿Cuál debe ser el cuerpo del bucle?**
    2. **¿Cuántas veces debe *iterarse* (ejecutarse) el cuerpo del bucle?**

Estructuras de repetición: bucles

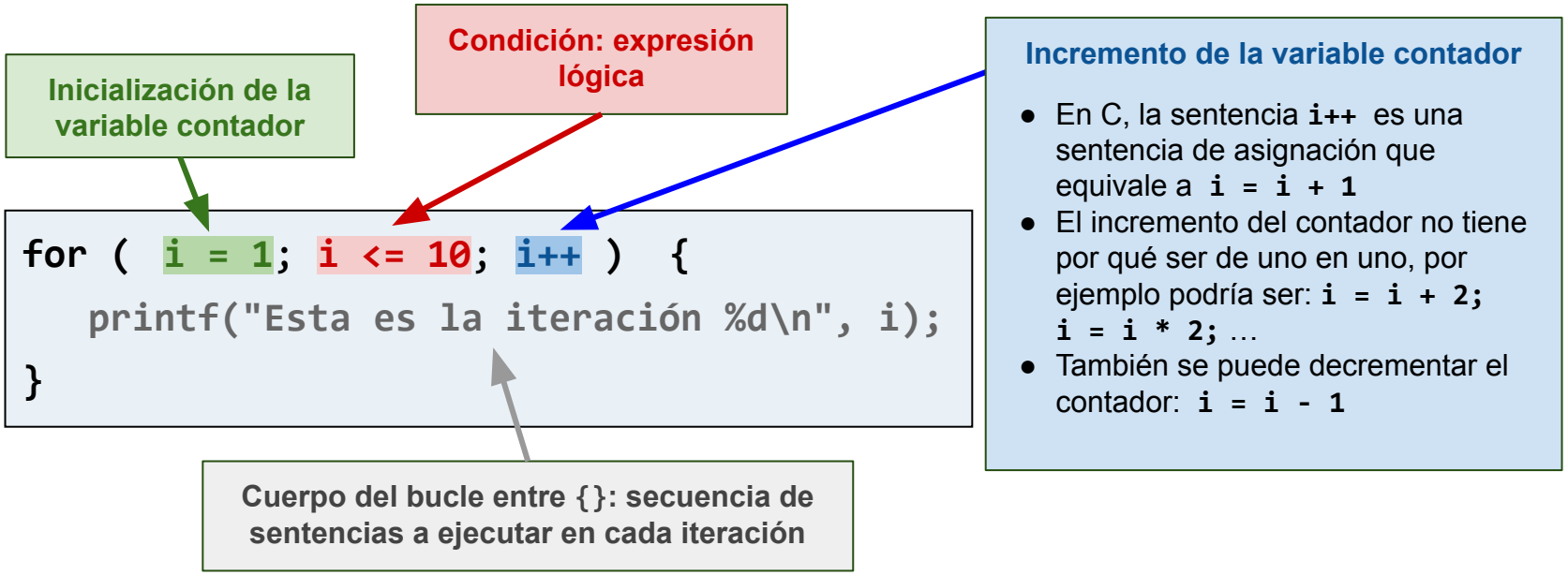
Tipos de bucles

- Dependiendo de si se conoce a priori, o no, el número de veces que se va a repetir el cuerpo del bucle (iteraciones), los bucles se pueden dividir en:
  - Bucles determinados. Antes de ejecutar el bucle se sabe el número de iteraciones.
    - Sentencia `for` (repetición con contador)
  - Bucles indeterminados. Antes de ejecutar el bucle no se sabe el número de iteraciones. El número de iteraciones final dependerá del cumplimiento de una condición.
    - Sentencia `while`
    - Sentencia `do ... while`

Estructuras de repetición: bucles determinados

Bucle `for`:

- Permite repetir un número determinado de veces (conocido a priori) la ejecución de una secuencia de sentencias.
- El número de iteraciones del bucle es controlado por una variable usada como un **contador**.



Estructuras de repetición: bucles determinados

Bucle `for`. ¿Cómo funciona?

Paso 1. Se ejecuta la sentencia de inicialización del contador (**una sola vez**).

Paso 2. Se evalúa la expresión lógica de forma que:

- Si su valor es **verdadero** Entonces se ejecuta el cuerpo del bucle.
- Si su valor es **falso** Entonces la sentencia `for` **FINALIZA** su ejecución.

Paso 3. Después de ejecutarse el cuerpo del bucle, se ejecuta la sentencia de incremento del contador.

Paso 4. Volver al **Paso 2**.

```
for ( i = 1;  i <= 10;  i++ )  {
   printf("Esta es la iteración %d\n", i);
}
```

Estructuras de repetición: bucles indeterminados

Bucle `while`:

- Permite repetir cero o más veces la ejecución de una secuencia de sentencias (el cuerpo del bucle) mientras la condición (expresión lógica) sea **verdadera**.

```
while (expresión_lógica) {
   secuencia de sentencias
}
```

```
caramelos = 0;
printf("¿Quieres un caramelo?: ");
scanf("%c", &res);
while (res == 'S' || res == 's') {
  caramelos = caramelos + 1;
  printf("¿Quieres otro caramelo?: ");
  scanf("%c", &res);
}  // fin de la sentencia while
printf("Te he dado %d caramelos\n", caramelos);
```

Estructuras de repetición: bucles indeterminados

Bucle do...while:

- Permite repetir una o más veces la ejecución de una secuencia de sentencias mientras la condición (expresión lógica) sea **verdadera**.

```
do {
   secuencia de sentencias
} while (expresión_lógica);
```

```
do {
  printf("Introduce opción (1-4): ");
  scanf("%d", &opcion);
} while (opcion < 1  ||  opcion > 4);
```

- Primero se ejecuta el cuerpo del bucle (secuencia de sentencias) y después se evalúa la expresión lógica.
- Mientras el resultado de evaluar *expresión_lógica* sea **verdadero** se ejecutará repetidamente el cuerpo del bucle.

Estructuras de repetición: bucles indeterminados

Equivalencia entre el bucle `for` y el bucle `while`

- Cualquier bucle **`for`** se puede reescribir como un bucle **`while`**

```
for (inic ; expr_log ; incr) {
   secuencia de sentencias;
}
```

```
for ( i = 1;  i <= 10; i++)  {
  printf("Esta es la iteración %d\n", i);
}
```

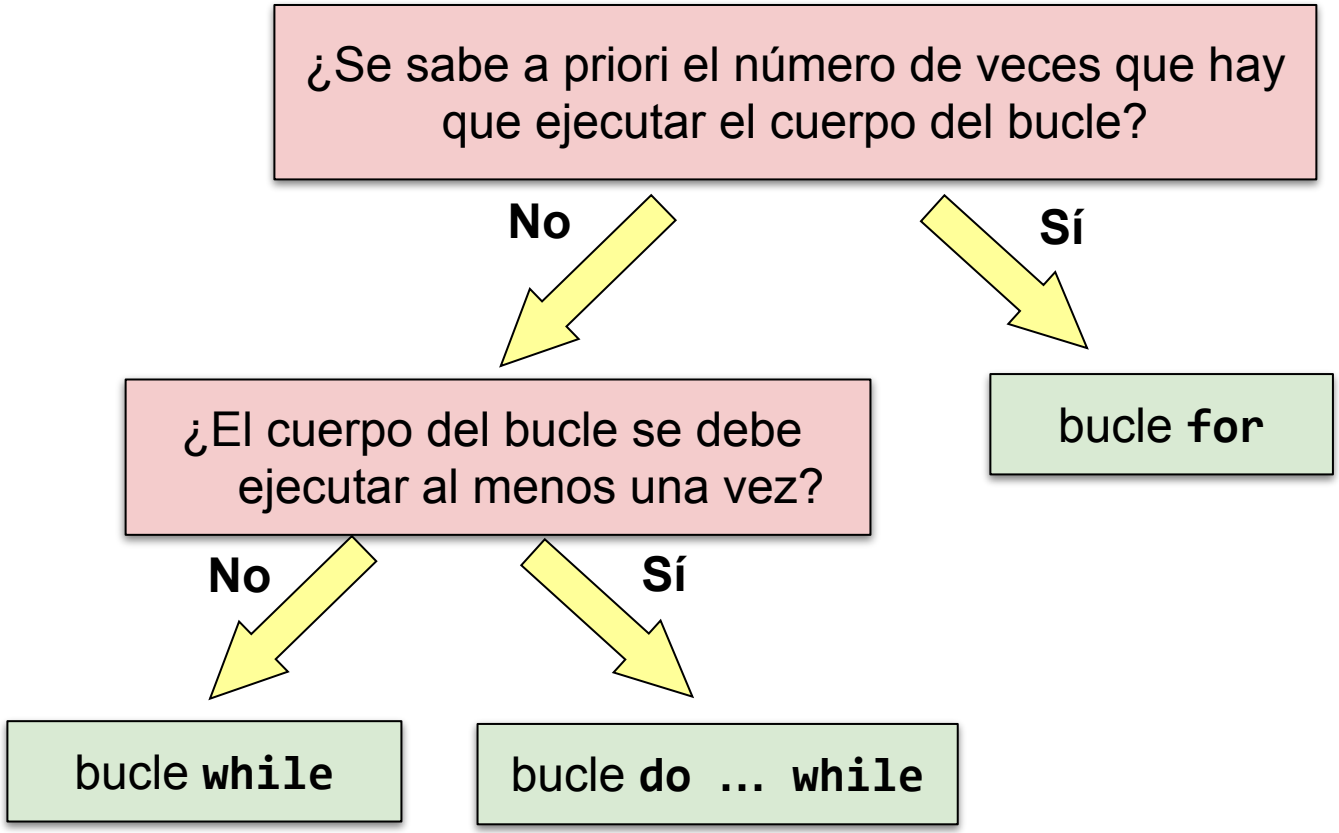
Se puede reescribir como

```
inic ;
while  (expr_log)  {
   secuencia de sentencias;
   incr;
}
```

```
i = 1;
while ( i <= 10 )  {
  printf("Esta es la iteración %d\n", i);
  i++;
}
```

Estructuras de repetición: bucles

Cómo saber qué tipo de bucle utilizar



4. Comentarios en el código fuente en C

- Son **notas aclaratorias** que podemos incluir en el código fuente un programa.
- Facilitan el mantenimiento del programa.
- Existen dos tipos de comentarios:
  - Comentario de línea. Se utiliza el símbolo `//` y todo lo que vaya a la derecha de ese símbolo en la misma línea es considerado como comentario por el compilador.

```
float nota_parcial;  // nota de un parcial (dato de entrada)
// Calcular la nota media e imprimirla por pantalla
```

  - Comentario de bloque. Se utilizan los símbolos `/*` y `*/`. Todo lo que vaya entre los símbolos es considerado como comentario por el compilador.

```
/* Introducir las notas de todos los parciales y sumarlas
(sólo cuando el dato introducido sea correcto) */
```

4. Comentarios en el código fuente en C

¿Cómo debe ser un comentario?

- Un comentario debe explicar de forma clara y concisa **qué** es lo que hace una sección del código de un programa, y **no cómo** lo hace (para eso ya está el propio código).

  Dado el siguiente código:

```
for (x = 1; x <= 10; x++)
  for (y = 1; y <= 10; y++)
    printf("%d * %d = %d\n", x, y, x*y);
```

  ¿Qué opinas del siguiente comentario para dicho código?

```
/* Tenemos 2 bucles for anidados que se repiten 10 veces cada uno. En el
bucle interno se imprime un mensaje de texto en la pantalla que indica el
producto de las dos variables usadas como contador en los bucles for. En
total se imprimen 100 líneas en la pantalla */
```

  ¿Cuál sería el comentario más adecuado?

4. Comentarios en el código fuente en C

¿Dónde hay que poner comentarios?

- En la definición de un módulo (qué hace el módulo).
- Al principio de una sección de código que realice una acción importante y no sea obvio lo que hace.
- Al principio del programa (una cabecera con el nombre del programa, autor, fecha, descripción del programa, etc.).

¿Cuántos comentarios hay que poner?

- Demasiados comentarios son tan malos como muy pocos.

5. Traza de un programa

- Es la **secuencia de estados** por los que pasa un programa, es decir, el valor que van tomando las variables a medida que se van ejecutando las sentencias del programa.
- La traza se lleva a cabo mediante la **ejecución manual**, y de forma secuencial, de las sentencias que componen el programa.
- Las trazas se utilizan principalmente para **depurar** un programa, es decir, para **corregir errores** detectados durante su ejecución.

5. Traza de un programa

Ejemplo de realización de una traza

```
// Dado un número N > 0, calcula la suma de
// todos los números impares menores que N
#include<stdio.h>

int main() {
  int num;  // número leído (dato de entrada)
  int suma; // resultado del sumatorio
            // (dato de salida)
  int i;    // contador de bucle (dato auxiliar)

  printf("Introduce un número > 0: ");
  scanf(" %d", &num);

  /* calcular el sumatorio e imprimirlo
     por pantalla */
  suma = 0;
  for (i=1; i < num; i++) {
     if ( (num % 2) != 0 )
        suma = suma + i;
  }
  printf("El resultado es: %d\n", suma);

  return 0;
}
```

	num	suma	i
scanf	5		
Inicializa suma	5	0	
Inicializa contador	5	0	1
1ª iteración del for	5	1	1
Incremento contador	5	1	2
2ª iteración del for	5	3	2
Incremento contador	5	3	3
3ª iteración del for	5	6	3
Incremento contador	5	6	4
4ª iteración del for	5	10	4
Incremento contador	5	10	5

¿Por qué no funciona?

5. Traza de un programa

Saber qué hace un programa mediante una traza

También se puede utilizar una traza para averiguar qué hace un programa o una parte del código del mismo.

```
#include<stdio.h>

int main() {
  float a, r;
  int b, i;

  printf("Introduce un número real: ");
  scanf(" %f", &a);
  printf("Introduce un número entero: ");
  scanf(" %d", &b);
  r = 1;
  for (i = 0; i < b; i++) {
    r = r * a;
  }

  printf("El resultado es: %.2f\n", r);

  return 0;
}
```

¿Qué hace este programa?

6. Estructura general de un programa


```
#directivas del preprocesador
Declaración de constantes


main() {
  Declaración de variables:
    de tipos simples
  Cuerpo principal (sentencias ejecutables)
    sentencias de Entrada y Salida
    sentencias de asignación
    estructuras de selección
    estructuras de repetición
}
```

6. Estructura general de un programa


```c
#include <stdio.h> // para usar sentencias de entrada/salida como printf y scanf
#include <stdbool.h> // para usar variables de tipo boolean

const int NUM_PARCIALES = 5;  // número de exámenes parciales

int main() {
  float nota_parcial;  // nota de un parcial (dato de entrada)
  float suma; // suma total de notas (dato auxiliar)
  int i;  // contador del bucle for (dato auxiliar)
  bool nota_incorrecta; // true si la nota introducida es incorrecta (dato auxiliar)
  float nota_final; // nota media de todos los parciales (dato de salida)

  suma = 0;

  // pedir todas las notas parciales y sumarlas cuando el dato introducido sea correcto
  for (i = 1; i <= NUM_PARCIALES; i++) {
    do {
      printf("Dime tu nota del parcial %d: ", i);
      scanf(" %f", &nota_parcial);
      nota_incorrecta = (nota_parcial < 0.0 || nota_parcial > 10.0);
      if (nota_incorrecta)
        printf("La nota introducida es incorrecta\n");
    } while (nota_incorrecta);
    suma = suma + nota_parcial;
  }

  // Calcular la nota media final e imprimirla por pantalla
  nota_final = suma / NUM_PARCIALES;
  printf("Tu nota final es: %.2f\n", nota_final);

  return 0;
}
```

Ejemplo de programa en C

7. Buenas prácticas para un código legible

- Los nombres de variables y constantes deben ser significativos.
- Utilizar correctamente la tabulación y saltos de línea entre partes que, por su lógica, deben considerarse por separado.
- Utilizar adecuadamente los comentarios en el código.
- Evitar estructuras `if-else` anidadas muy profundas.

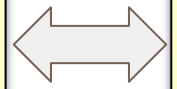
```
#include<stdio.h>

int main() {
  float base, potencia;
  int exponente, i;

  printf("Introduce un número real: ");
  scanf(" %f", &base);
  printf("Introduce un número entero: ");
  scanf(" %d", &exponente);
  potencia = 1;
  for (i = 0; i < exponente; i++)
    potencia = potencia * base;

  printf("El resultado es: %.4f\n", potencia);

  return 0;
}
```



```
#include<stdio.h>

int main() {   float a, r;   int
 b, i;   printf("Introduce un  número
real: ";  scanf(" %f", &a);
printf(
"Introduce un número entero: ");
  scanf(" %d", &b);   r
 = 1;   for
(i=0
; i < b;
i++)       r = r * a;
  printf( "El resultado es: %f\n",
r);
     return 0;}
```

Los dos programas hacen lo mismo, pero un programa escrito con un buen estilo de programación es más fácil de leer (más legible) y más fácil de modificar (más mantenible)

Ejercicios

5. Después de ejecutar cada uno de los siguientes fragmentos de programa, ¿cuál será el valor final de la variable *x* en cada uno de los casos?

Caso A

```
x = 0;
n = 16;
while ( n != 0) {
    x = x + n;
    n = n / 2;
}
```

Caso B

```
z = 12;
x = 0;
if ((z % 4) == 0)
   for (j = 0; j < 10; j + 4)
       x = x + j;
else
    for (j = 0; j < 10; j + 2)
       x = x + j;
```

6. Escribe un programa que lea números positivos y nos muestre el valor de su suma y la cantidad de números leídos. Finaliza cuando se introduce un número negativo.

7. Escribe un programa que lea un número entero mayor que cero y construya y visualice otro número formado por las mismas cifras pero en sentido contrario.

8. Modifica el programa del ejercicio 4 para añadir una cuarta opción que sea SALIR. El programa debe mostrar el menú continuamente, después de que el usuario elija opción, hasta que se elija la opción 4.