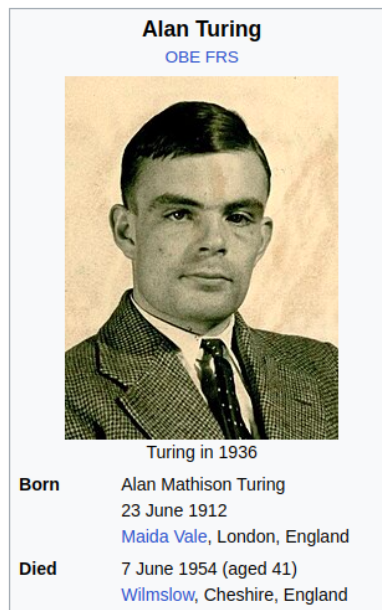


(Versión de fecha 22 de octubre de 2024)

## 1 LENGUAJES TURING COMPLETOS



LENGUAJE TURING COMPLETO: permite resolver cualquier algoritmo de computación.

- 1- Operador de asignación. Variables.
- 2- Bifurcaciones lógicas.
- 3- Bucles (Repeticiones).

Figura 1: Condiciones que tiene que reunir un lenguaje para ser Turing completo, esto es, para poder resolver cualquier algoritmo de computación

## 2 CONCEPTO DE VARIABLE. OPERADOR DE ASIGNACIÓN

Una *variable* es un identificador (etiqueta) que se asigna a un valor determinado que está guardado en la memoria. Cuando se quiere acceder al valor, se utiliza el nombre de la variable.

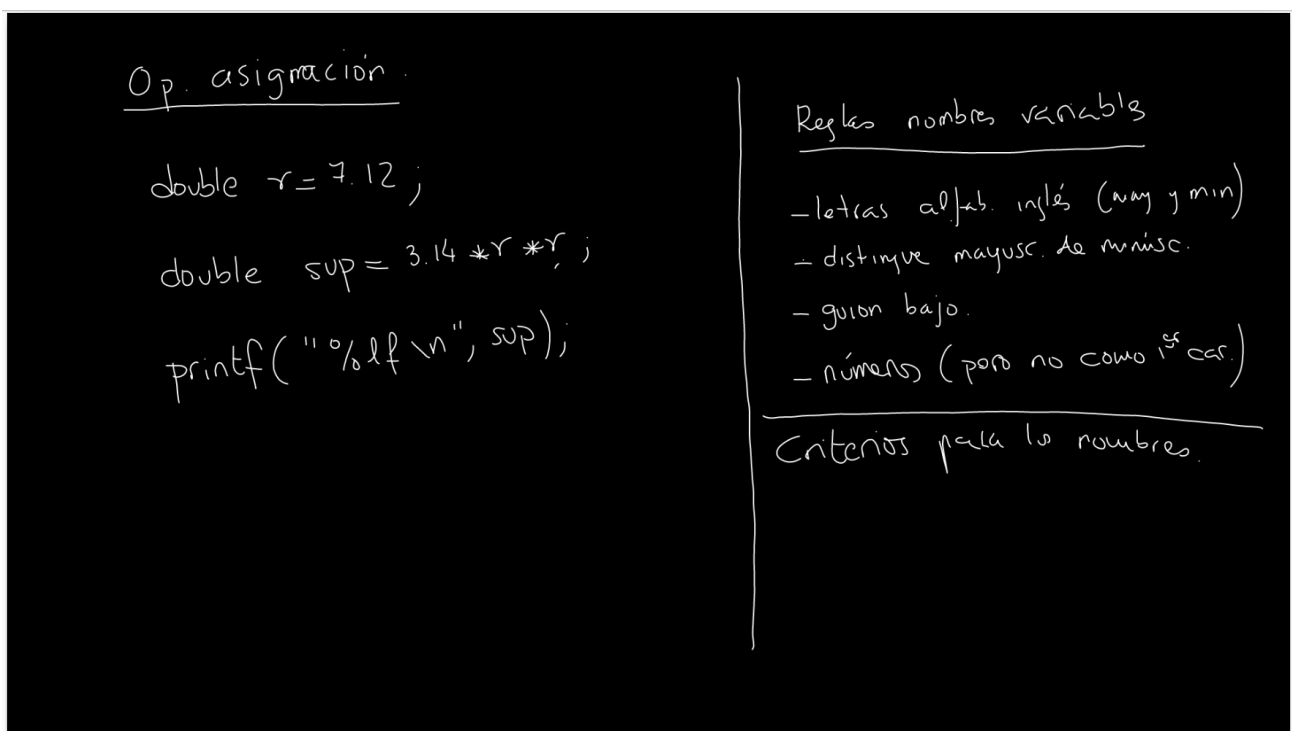
Para utilizar las variables hay que proceder en dos pasos:

- **Declaración:** para utilizar una variable hay que declararla, o sea, decirle al compilador el tipo de datos del valor que va a guardar la variable y el nombre que se le va a asignar. La sintaxis es la siguiente:

```
tipo_datos nombre_variable;
```

- **Asignación:** el operador de asignación es el signo igual (=). Asigna a la variable cuyo nombre aparece a la izquierda del signo igual el resultado de la expresión que aparece a la derecha del signo igual.

La declaración y la asignación se pueden hacer en una misma línea de código, aunque siguen siendo dos instrucciones.



### Para saber más: Criterios para nombres de identificadores



(Imagen obtenida de la Wikipedia)

Hay varios criterios habituales para dar nombre a los identificadores de las variables, las funciones, las clases y otras construcciones de los lenguajes de programación:

- **camelCase:** si el identificador tiene una sola palabra, se pone en minúsculas. Si tiene más de una palabra, la primera se pone en minúsculas y las demás en mayúsculas, si espacios ni guiones bajos de separación. En Java se utiliza para los nombres de las variables, de las funciones y de los paquetes. También se llama a veces *Lower Camel Case*. Los siguientes serían ejemplos de identificadores utilizando el criterio *camelCase*:

valorInicial      getName()      x0

- **PascalCase:** es como el *camelCase*, pero poniendo todas las palabras en mayúsculas. En Java se utiliza para los nombres de las clases y de los interfaces. También se llama a veces *Upper Camel Case* o simplemente *CamelCase* (con la primera C mayúscula). Los siguientes serían ejemplos de identificadores utilizando el criterio *PascalCase*:

Animal      VehiculoElectrico      Iterable

- **snake\_case:** en este convenio se utilizan palabras en minúsculas separadas por el guion bajo. No es frecuente utilizarlo en Java, pero sí en otros lenguajes, como Rust. Los siguientes serían ejemplos de identificadores utilizando el criterio *snake\_case*:

valor\_inicial      get\_name()      x\_0

- **SCREAMING\_SNAKE\_CASE:** es como el *snake\_case*, pero poniendo las palabras con todas las letras mayúsculas. Se suele utilizar en todos los lenguajes para nombrar valores constantes. Los siguientes son algunos ejemplos de este criterio:

VALOR\_INICIAL      PI      X\_0

### 3 BIFURCACIONES IF



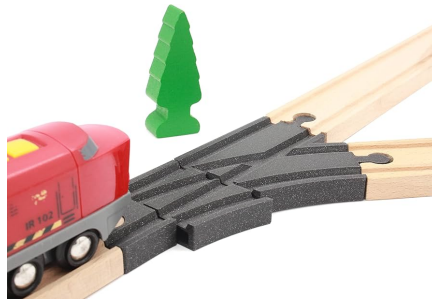
Bifurcaciones.

```
int n = 10;  
int result = n;  
if (n < 0) {  
    result = result * (-1);  
}  
printf("%d\n", result);
```

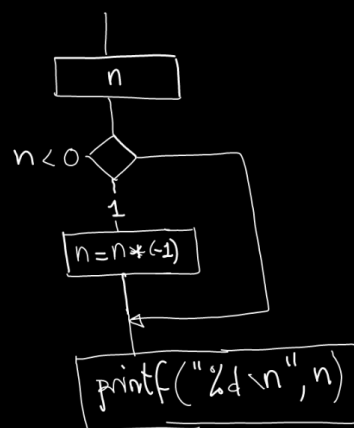
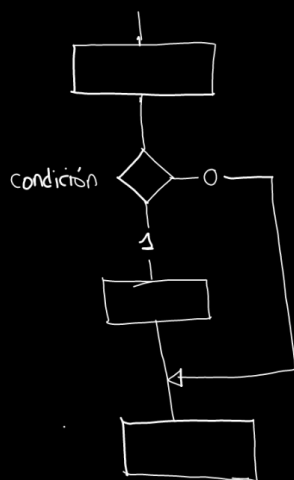
```
int n = 11;  
if (n % 2 == 0) {  
    printf("PAR");  
} else {  
    printf("IMPAR");  
}
```

```
char talla;  
int medida = 43;  
if (medida > 60) {  
    talla = 'L';  
} else if (medida > 50) {  
    talla = 'M';  
} else if (medida > 40) {  
    talla = 'S';  
} else {  
    talla = 'X';  
}  
↓  
✗
```

## 4 BIFURCACIÓN IF SIMPLE (SIN RAMA ELSE)



### BIFURCACIÓN SIMPLE



```
int n = 10;  
if (n < 0) {  
    n = n * (-1);  
}  
printf("%d \n", n);
```

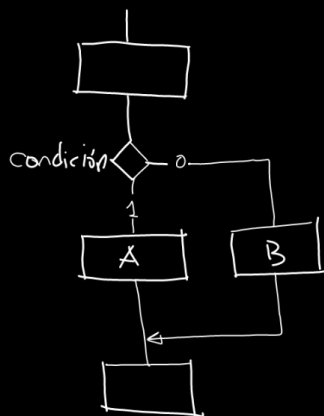
### Ejemplo 1 Ejemplo de bifurcación simple: cálculo del valor absoluto

```
#include <stdio.h>  
  
int main() {  
    // Inicialización  
    printf("Calcula el valor absoluto de un número");  
    int n;  
  
    // Entrada  
    printf("n=");  
    scanf("%d", &n);  
  
    // Procesamiento  
    if(n < 0) {  
        n = n * (-1);  
    }  
  
    // Salida  
    printf("El valor absoluto es: %d \n", n);  
}
```

## 5 BIFURCACIÓN IF CON RAMA ELSE



### BIFURCACIÓN if... else

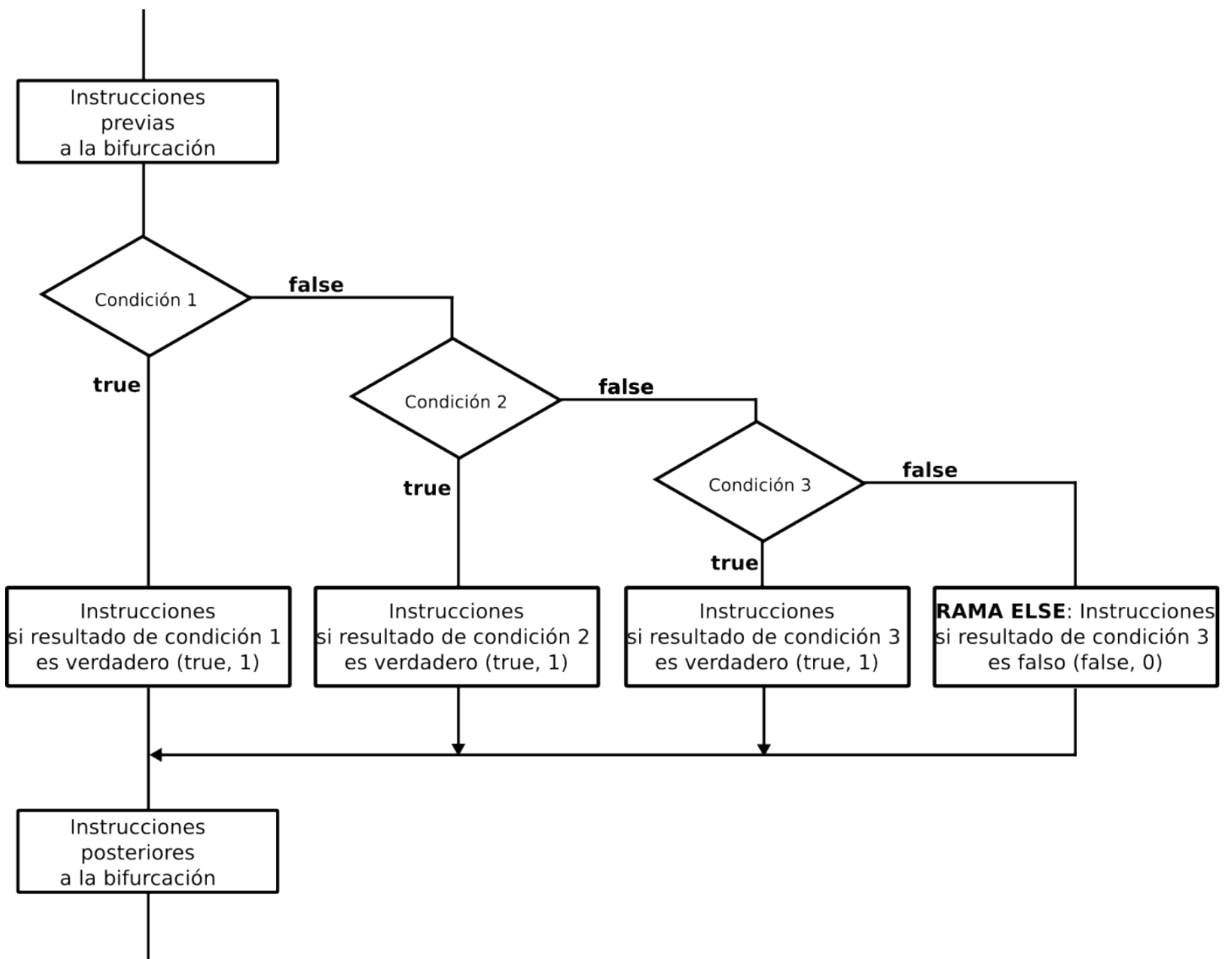


```
int n;  
scanf("%d", &n);  
  
if (n%2 == 0) {  
    A  
} else {  
    B  
}
```

### Ejemplo 2 Ejemplo de bifurcación con rama else: cálculo de la paridad

```
#include <stdio.h>  
  
int main() {  
    // Inicialización  
    printf("Calcula par o impar\n");  
    int n;  
  
    // Entrada  
    printf("n= ");  
    scanf("%d", &n);  
  
    // Procesamiento y salida  
    if(n%2 == 0) {  
        printf("PAR\n");  
    } else {  
        printf("IMPAR\n");  
    }  
    printf("FIN\n");  
}
```

## 6 BIFURCACIÓN MULTICONDICIONAL: if...else if... else



### Ejemplo 3 Bifurcación multicondicional if ... else if ... else

```
#include <stdio.h>

int main( void ){
    char talla;
    int medida;

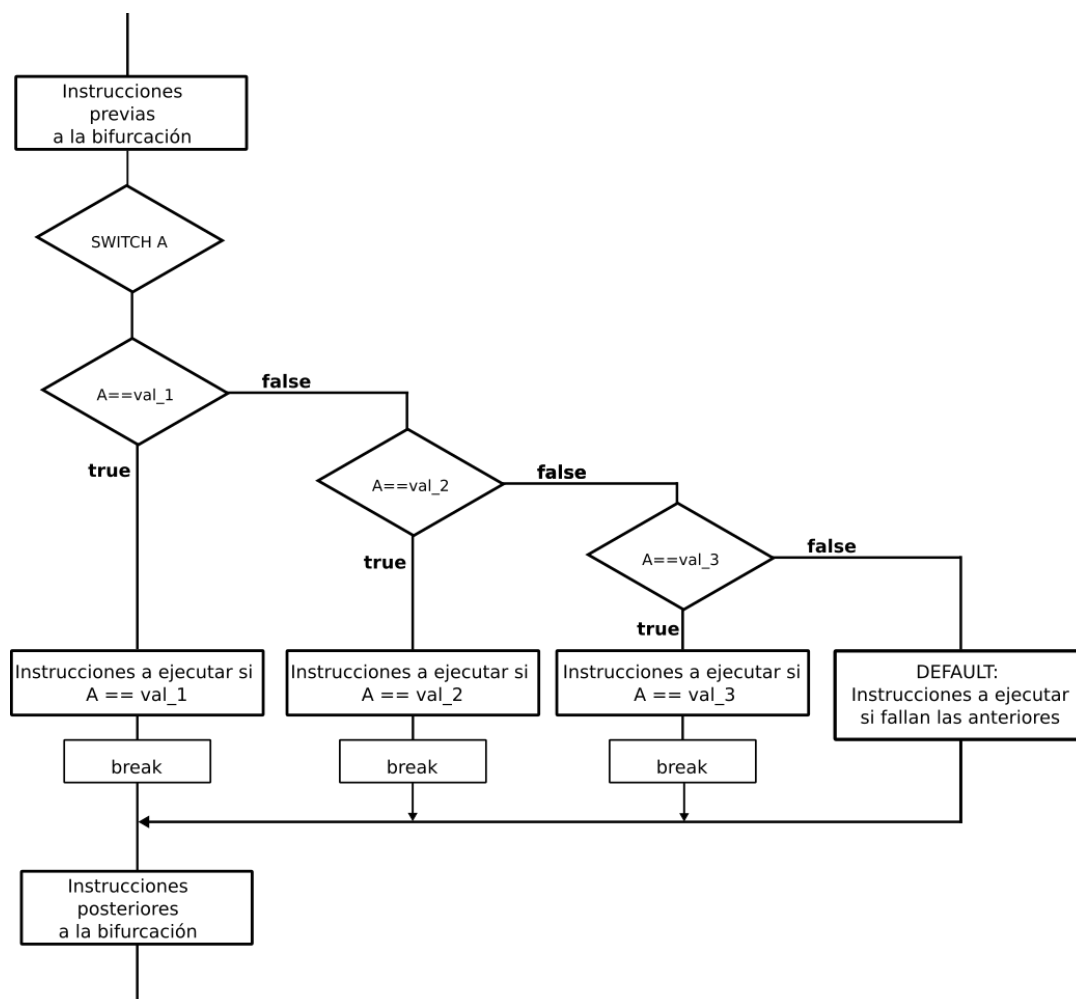
    printf("Medida: ");
    scanf("%d", &medida);

    if(medida>60) {
        talla = 'L';
    } else if(medida>50) {
        talla = 'M';
    } else if(medida>40) {
        talla = 'S';
    } else {
        talla = 'X';
    }

    printf("Talla: %c\n", talla);

    return 0;
}
```

## 7 BIFURCACIÓN SWITCH...CASE



#### Ejemplo 4 Ejemplo switch ... case

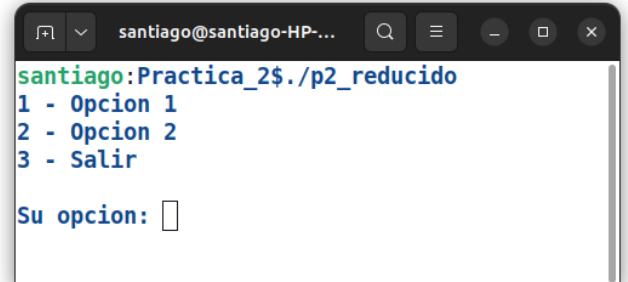
```
#include <stdio.h>

int main() {

    int opcion = 0;
    while(opcion != 3) {
        // Mostrar menú
        printf("1 - Opcion 1\n");
        printf("2 - Opcion 2\n");
        printf("3 - Salir\n");

        // Leer opcion
        printf("\nSu opcion: ");
        scanf("%d", &opcion);

        // Bifurcar
        switch(opcion) {
            case 1:
                printf("Ejecutando opcion 1\n");
                break;
            case 2:
                printf("Ejecutando opcion 2\n");
                break;
            case 3:
                printf("Adios\n");
                break;
            default:
                printf("Opción incorrecta\n");
                break;
        }
    }
}
```



```
santiago@santiago-HP-...
santiago:Practica_2$./p2_reducido
1 - Opcion 1
2 - Opcion 2
3 - Salir

Su opcion: 
```



## 8 BUCLES

### Bucles

```
double x, y, z;  
scanf("%lf", &x);  
scanf("%lf", &y);  
scanf("%lf", &z);
```

Dos tipos de bucles:

- while : while() do { while()
- for

```
int n;  
scanf("%d", &n); //user's token 3  
int contador = 0;
```

```
while (contador < n) {  
    printf("%d \n", contador);  
    contador = contador + 1;  
}
```

```
printf("Fin \n");
```

0 ↴  
1 ↴  
2 ↴  
Fin ↴

```
do {
    [ ]
} while (condicion);
```

↓

x

```
int n;
int condicion = 1;

while (condicion == 1) {
    scanf("%d", &n);
    if (n == -1) {
        condicion = 0;
    }
}
```

↓

```
3 ↗
5 ↘
-1 ↘
```

## 9 BUCLES *WHILE* Y *DO...WHILE*

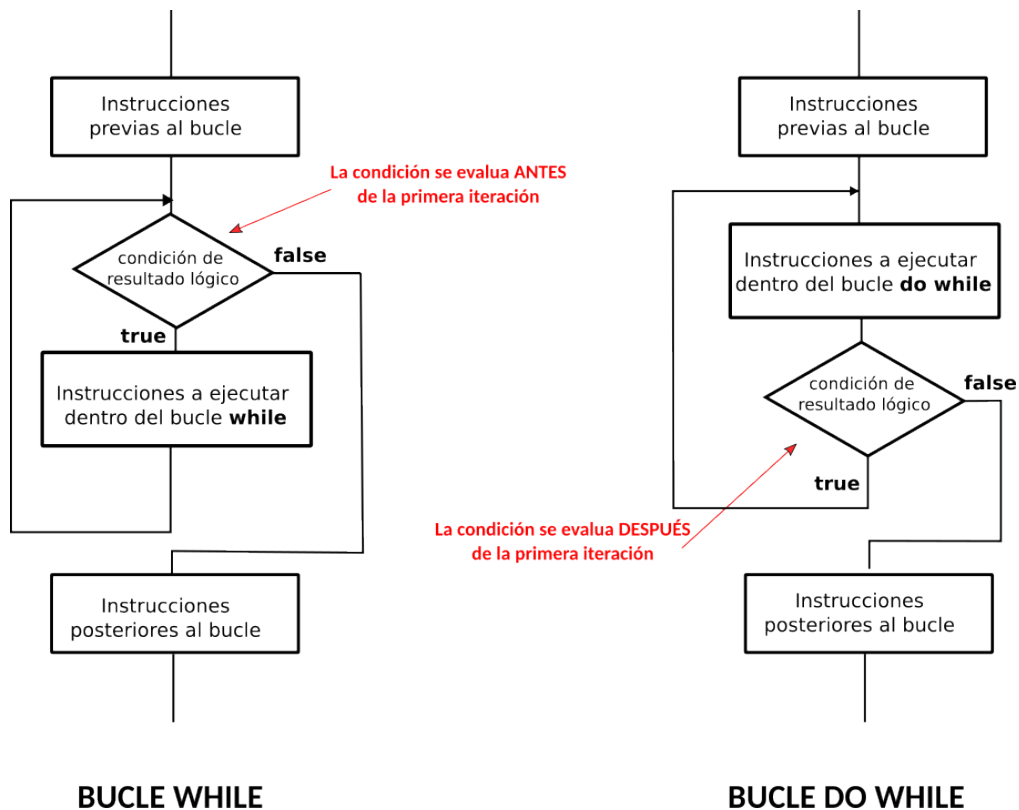


Figura 2: Izquierda: diagrama de flujo de un bucle *while*. Derecha: diagrama de flujo de un bucle *do ... while*

El siguiente ejemplo muestra en pantalla los números del 0 al 9, la primera vez utilizando un bucle *while* y la segunda vez mediante un bucle *do while*.

### Ejemplo 5 Bucles *while* y *do while*

```
#include <stdio.h>

int main( void ){
    int num = 0;

    while(num<10) {
        printf("%d ", num);
        num = num + 1;
    }
    printf("\n");

    num = 0;
    do {
        printf("%d ", num);
        num = num + 1;
    } while(num<10);
    printf("\n");

    printf("Pulse una tecla para finalizar\n");
    getchar();
    return 0;
}
```

La imagen muestra una ventana de terminal con el título 'shig...'. El prompt de shell es '\$> ./ej\_6'. La salida del programa muestra dos líneas de números del 0 al 9, separados por espacios, una para cada tipo de bucle. Después de la segunda línea, aparece el mensaje 'Pulse una tecla para finalizar' y un cursor parpadeante en la línea siguiente.

## 10 BUCLES FOR

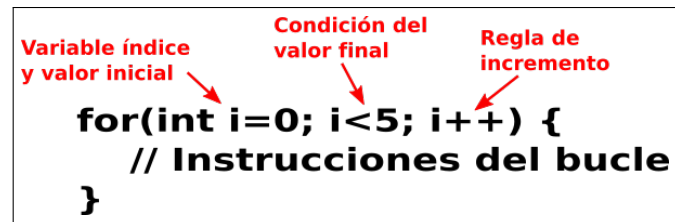


Figura 3: Sintaxis de la instrucción *for*

El siguiente ejemplo muestra en pantalla los números pares desde el 0 hasta el 20.

### Ejemplo 6 Ejemplo de bucle *for*

```
#include <stdio.h>  
  
int main(){  
    for(int i=0; i<10; i=i+2) {  
        printf("%d ", i);  
    }  
    printf("\n");  
  
    getchar();  
    return 0;  
}
```

Captura de pantalla de una terminal ejecutando el programa. La ventana de la terminal tiene el título "shig...". El prompt es "\$>". Se ha ejecutado el comando `./ej_7`, lo que ha resultado en la salida de los números pares desde 0 hasta 20: `0 2 4 6 8 10 12 14 16 18 20`. El cursor se encuentra en la línea siguiente.

## 11 OPERADORES UNARIOS

### Operadores unarios

$+=$     $*=$

`int n=10;`

`n+=1;`  $\rightarrow n=n+1;$

$++$

`n++;`  $\rightarrow n=n+1;$

`++n;`

`x = n++;`  $\left\{ \begin{array}{l} x=n; \\ n=n+1; \end{array} \right.$

`x = ++n;`  $\left\{ \begin{array}{l} n=n+1; \\ x=n; \end{array} \right.$

### Ejemplo 7 Ejemplo operadores unarios

```
#include <stdio.h>

int main() {

    int n = 0;

    n++; // Ahora n vale 1

    n+=2; // Ahora n vale 3

    // Post incremento: primero asigna y luego incrementa
    int p = n++; // p vale 3, n vale 4

    // Pre incremento: primero incrementa y luego asigna
    int q = ++n; // q vale 5, n vale 5

}
```

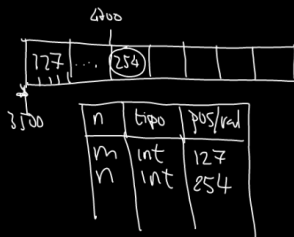
## 12 TIPOS DE DATOS COMPUESTOS

### TIPOS DE DATOS COMPUESTOS

- ARRAYS: varios elem. del mismo tipo  
ordenados — acceso por índice
- ESTRUCTURAS: varios elem que pueden ser de distinto tipo  
acceso por punto

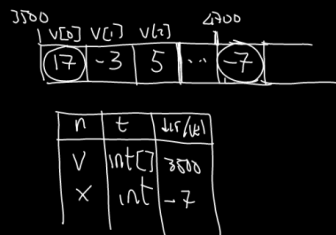
### ARRAYS

```
int m;  
m = 127;  
int n = 2 * m;
```



```
int v[3];  
v[0] = 17;  
v[1] = -3;  
v[2] = 5;  
int x = 2 * v[2] - v[0];
```

tamaño = 3  
último índice = 2  
primer índice = 0



```

double x[] = { 2.5, 3.2, -0.5 };
double x[3] = { 2.5, 3.2, -0.5 };

scanf("%lf", &x[1]);
printf("(%.2lf, %.2lf, %.2lf)\n", x[0], x[1], x[2]);

double x[3] = { };
double x[3] = { 0 };

```

```

int n;
scanf("%d", &n);
double x[n];

```

---

```

int p[3];
p = { ..., ..., ... };


```

```

#define DIM 10 3.14
          3.14
          PI

```

```

int main() {
    int valores[DIM];
    3.14
    PI
    }

```

size\_t

## Ejemplo 8 Declaración y asignación de arrays

```
#include <stdio.h>

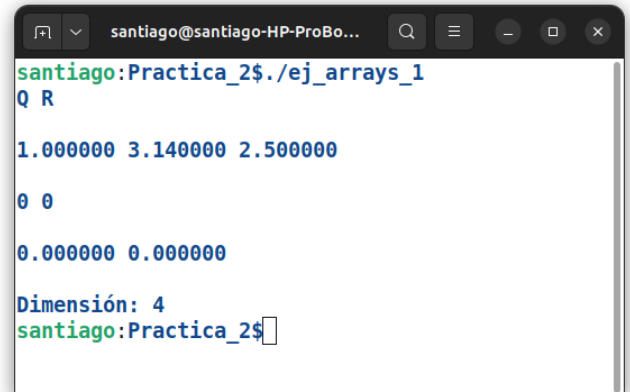
int main() {
    // Declaración y asignación en distintas líneas
    char cars[2];
    cars[0] = 'Q';
    cars[1] = 'R';
    printf("%c %c\n\n", cars[0], cars[1]);

    // Declaración y asignación en la misma línea
    double x[] = {1.0, 3.14, 2.5};
    printf("%lf %lf %lf\n\n", x[0], x[1], x[2]);

    // Inicialización a 0 usando llaves vacías
    int v[2] = {};
    printf("%d %d \n\n", v[0], v[1]);

    // Inicialización a 0 poniendo un 0 entre llaves
    double w[2] = {0};
    printf("%lf %lf \n\n", w[0], w[1]);

    // Inicialización dinámica de un array
    int dim;
    printf("Dimensión: ");
    scanf("%d", &dim);
    double y[dim];
}
```



```
santiago:Practica_2$ ./ej_arrays_1
Q R

1.000000 3.140000 2.500000

0 0

0.000000 0.000000

Dimensión: 4
santiago:Practica_2$
```

## Ejemplo 9 Imprimir array, 1 elemento por línea

```
#include <stdio.h>

#define DIM 3

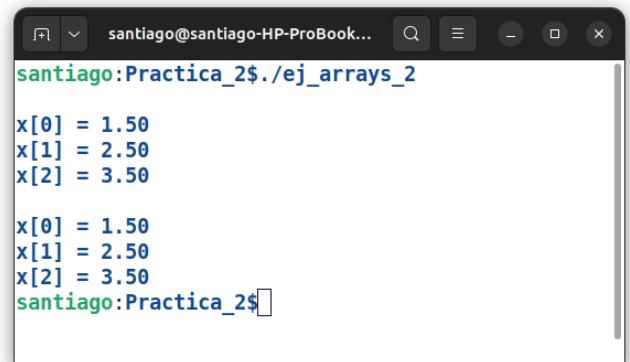
int main() {

    double x[DIM] = {1.5, 2.5, 3.5};

    // Imprimir con bucle for, 1 elem por línea
    for(int i=0; i<DIM; i++) {
        printf("x[%d] = %.2f\n", i, x[i]);
    }

    printf("\n");

    // Imprimir con while, 1 elem por línea
    int n = 0;
    while(n<3) {
        printf("x[%d] = %.2f\n", n, x[n]);
        n++;
    }
}
```



```
santiago:Practica_2$ ./ej_arrays_2
x[0] = 1.50
x[1] = 2.50
x[2] = 3.50

x[0] = 1.50
x[1] = 2.50
x[2] = 3.50
santiago:Practica_2$
```



## Ejemplo 10 Imprimir array en una sola línea

```
#include <stdio.h>

#define DIM 3

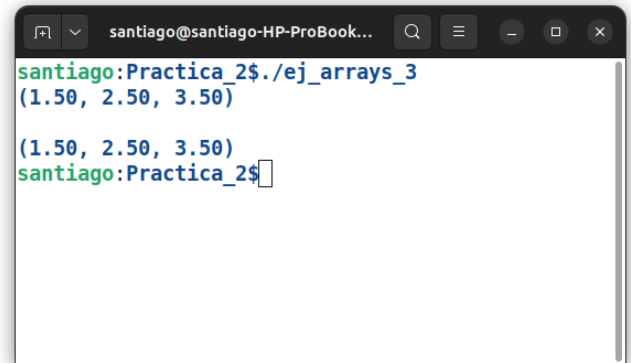
int main() {

    double x[DIM] = {1.5, 2.5, 3.5};

    // Imprimir en una línea usando un if
    printf("(");
    for(int i=0; i<DIM; i++) {
        printf("%.2f", x[i]);
        if(i<DIM-1) {
            printf(", ");
        }
    }
    printf(")\n");

    printf("\n");

    // Imprimir en una línea usando backspace
    printf("(");
    for(int i=0; i<DIM; i++) {
        printf("%.2f, ", x[i]);
    }
    printf("\b\b)\n");
}
```



```
santiago:Practica_2$ ./ej_arrays_3
(1.50, 2.50, 3.50)
(1.50, 2.50, 3.50)
santiago:Practica_2$
```