Cultura Digital 1

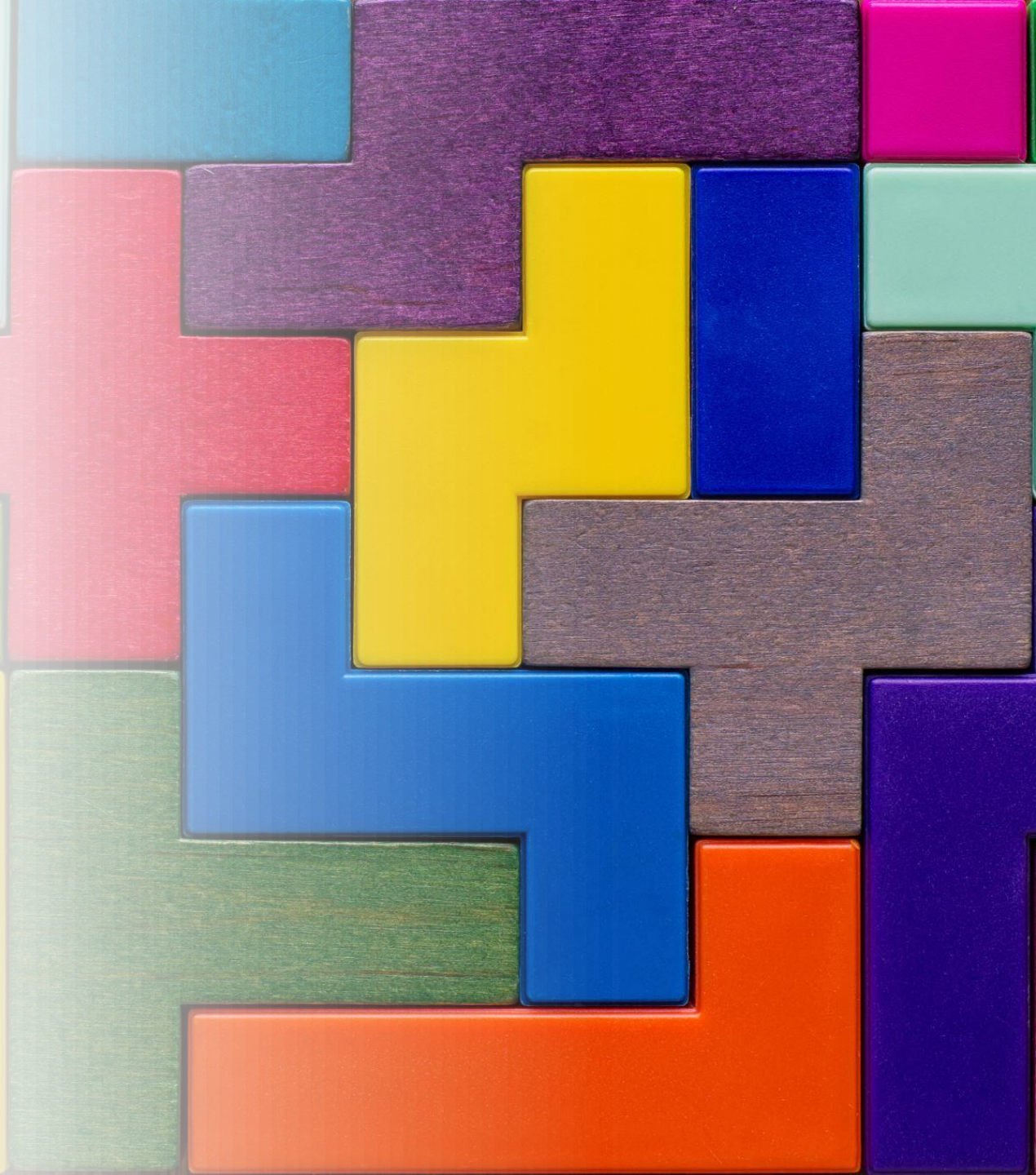
Pensamiento Computacional y Lenguaje Algorítmico

BACHILLERATO VIRTUAL

COBATAB

Autor: Mtro. Mario Alejandro Rodríguez Ramón

Julio 2023



Conceptos básicos

Variables, Operadores, Constantes, Palabras reservadas

Variables y Constantes

Los programas de computadoras utilizan diferentes tipos de datos, por lo que requieren de algún mecanismo para almacenar el conjunto de valores usado. El lenguaje C ofrece dos posibilidades: *variables* y *constantes*.

Una variable es un objeto donde se guarda un valor, el cual puede ser consultado y modificado durante la ejecución del programa. Por el contrario, una constante tiene un valor fijo que no puede ser modificado.

Ejemplos de variables

Toda variable debe declararse antes de ser usada por primera vez en el programa. Las sentencias de declaración de variables indican al compilador que debe reservar cierto espacio en la memoria del ordenador con el fin de almacenar un dato de tipo elemental o estructurado.

La declaración consiste en dar un nombre significativo a la variable e indicar el tipo de datos a que corresponden los valores que almacenará. A continuación se muestra la sintaxis más sencilla de una sentencia de declaración para una sola variable:

```
tipo datos nombre variable;  ⟶  int a;
```

Ejemplo de constantes

Constantes literales

Todo valor que aparece directamente en el código fuente cada vez que es necesario para una operación constituye una *constante literal*. En el siguiente ejemplo, los valores 20 y 3 son constantes literales del tipo de datos entero:

```
int cont = 20;

cont = cont + 3;
```

Constantes simbólicas

Una *constante simbólica* es una constante representada mediante un nombre (símbolo) en el programa. Al igual que las constantes literales, no pueden cambiar su valor. Sin embargo para usar el valor constante, se utiliza su nombre simbólico, de la misma forma que lo haríamos con una variable. Una constante simbólica se declara una sola vez, indicando el nombre y el valor que representa.

El método más habitual para definir constantes en C es la directiva del preprocesador #define. Por ejemplo:

```
#define PI 3.14159
```

Identificadores

- Un *identificador* en un lenguaje de programación es un nombre utilizado para referir un valor constante, una variable, una estructura de datos compleja, o una función, dentro de un programa. Todo identificador está formado por una secuencia de letras, números y caracteres de subrayado, con la restricción de que siempre debe comenzar por una letra o un subrayado y que no puede contener espacios en blanco.
- Cada compilador fija un máximo para la longitud de los identificadores, siendo habitual un máximo de 32 caracteres. C diferencia entre mayúsculas y minúsculas, según lo cual C considerará los identificadores contador, Contador y CONTADOR, por ejemplo, como diferentes. En cualquier caso, nunca pueden utilizarse las *palabras reservadas* del lenguaje para la construcción de identificadores. De acuerdo con el estándar ANSI, C consta ´únicamente de 32 palabras reservadas

Palabras reservadas

Palabras reservadas de C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Operadores

Un operador es un símbolo (+, -, *, /, etc.) que tiene una función predefinida (suma, resta, multiplicación, etc.) dentro de una expresión (normalmente una fórmula matemática) del programa.

Prioridad y asociatividad de los operadores en C

Operador	Símbolo	Asociatividad
Paréntesis	()	Izquierda a derecha
NO lógico Signo negativo Incremento Decremento	! − ++ −−	Derecha a izquierda
Multiplicación División Módulo	* / %	Izquierda a derecha
Suma Resta	+ −	Izquierda a derecha
Menor que Menor o igual que Mayor que Mayor o igual que	< <= > >=	Izquierda a derecha
Igual que Distinto que	== !=	Izquierda a derecha
Y lógico	&&	Izquierda a derecha
O lógico	||	Izquierda a derecha
Asignaciones	= += −= *= /= %=	Derecha a izquierda

Flujo de Control 1

Estructuras selectivas

Estructuras selectivas

- Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también *estructuras de decisión o alternativas*.
- En las estructuras selectivas se evalúa una condición y en función del resultado de la misma se rea liza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (**if**, **then**, **else** o bien en español **si**, **entonces**, **si_no**), con una figura geométrica en forma de rombo o bien con un triángulo en el interior de una caja rectangular. Las estructuras selectivas o alternativas pueden ser:
  - *simples,*
  - *dobles,*
  - *múltiples*.

Selectiva simple

Diagrama de sintaxis

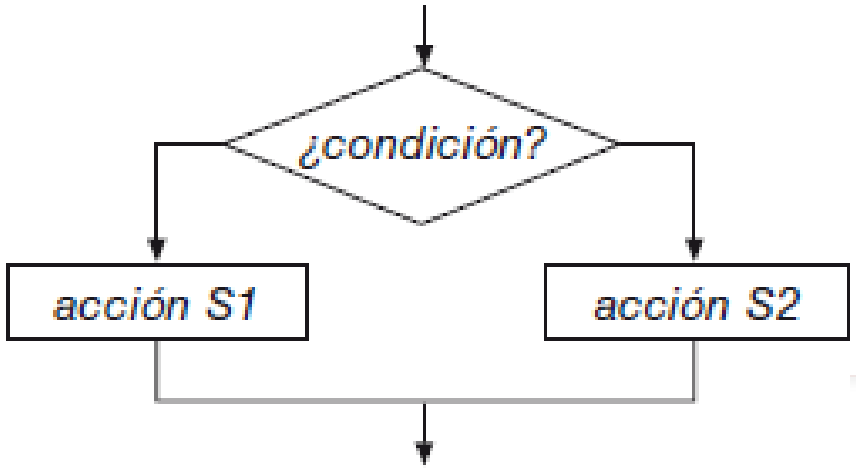
Sentencia **if_simple::=**

1. si (<expresión_lógica>)
   inicio
      <sentencia>
   fin

2. si <*expresión lógica*> entonces
      <*Sentencia_compuesta*>
   fin-si

Sentencia_compuesta ::=
   inicio
      <*Sentencias*>
   fin

Alternativa doble



a)

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición. Si la condición C es verdadera, se ejecuta la acción S1 y, si es falsa, se ejecuta la acción S2

Pseudocódigo en español

```
si <condicion> entonces
   <accion S1>
si _ no
   <accion S2>
fin _ si
```

Pseudocódigo en inglés

```
if <condicion> then
   <accion S1>
else
   <accion S2>
endif
```

Alternativa múltiple

La estructura de decisión múltiple evaluará una expresión que podrá tomar *n* valores distintos, 1, 2, 3, 4, ..., *n*. Según que elija uno de estos valores en la condición, se realizará una de las *n* acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los *n* posibles.

Figura 4.6. Estructuras de decisión múltiple.

```
Modelo 6:

según_sea (expresión) hacer
   caso expresión constante :
      [Sentencia
       sentencia
       ...
       sentencia de ruptura | sentencia ir_a ]
   caso expresión constante :
      [Sentencia
       sentencia
       ...
       sentencia de ruptura | sentencia ir_a ]
   caso expresión constante :
      [Sentencia
       ...
       sentencia
       sentencia de ruptura | sentencia ir_a ]
   [otros:
      [Sentencia
       ...
       sentencia
       sentencia de ruptura | sentencia ir_a ]
fin_según
```

Flujo de Control 2

Estructuras repetitivas

Estructuras repetitivas

Las estructuras de control repetitivas o iterativas permiten la ejecución repetida de instrucciones o sentencias de acuerdo a un número predeterminado de veces o dependiendo de cierta condición.

Las estructuras repetitivas más usuales son:

- Mientras (while)
- Hacer – mientras (Do-while)
- Desde/Para (For)

Estructura "mientras" (while)

La construcción `while` es similar a la existente en otros lenguajes de programación. Sin embargo, debido a que en C toda sentencia puede considerarse como una expresión, la construcción `while` de C ofrece cierta potencia añadida.

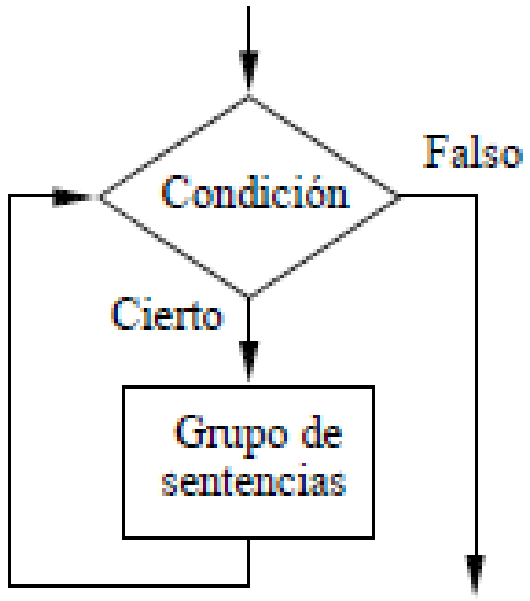
La forma más general para la ejecución repetida de una sola sentencia es:

```
while (condición)
   sentencia;
```

O para la ejecución repetida de un grupo de sentencias:

```
while (condición)
{
   grupo_de_sentencias;
}
```

El funcionamiento de esta construcción es bastante simple. El *cuerpo del bucle*, es decir, la sentencia o grupo de sentencias dentro del bucle, se ejecuta mientras el valor de la expresión que actúa de condición sea cierto. En el momento en que la condición sea falsa, la ejecución del programa continúa secuencialmente con la siguiente instrucción tras el bucle (ver Fig. 5.1).



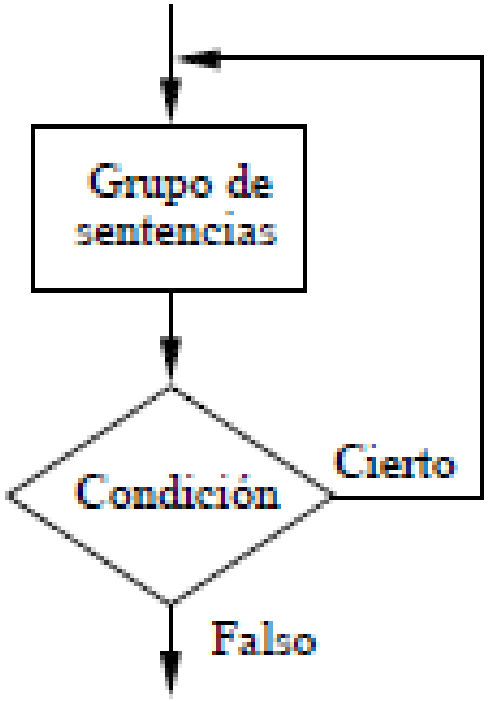
Estructura "hacer – mientras" (do - while)

La forma general de la construcción `do-while` es la siguiente:

```
do
{
   sentencia;   o   grupo_de_sentencias;
} while (condición);
```

Nótese que tanto para ejecutar una sola sentencia como para ejecutar un grupo de ellas, las llaves { } son igualmente necesarias.

Esta construcción funciona de manera muy similar a la construcción `while`. Sin embargo, al contrario que ésta, `do-while` ejecuta primero el cuerpo del bucle y después evalúa la condición. Por lo cual, el cuerpo del bucle se ejecuta como mínimo 1 vez (ver Fig. 5.2).



Estructura “desde/para” (for)

Esta construcción iterativa no presenta un formato fijo estricto, sino que admite numerosas variantes, lo que la dota de gran potencia y flexibilidad.

Su forma más general para la ejecución repetida de una sola sentencia es:

```
for (sentencia_inicial ; condición ; incremento/decremento)
   sentencia;
```

o para la ejecución repetida de un grupo de sentencias:

```
for (sentencia_inicial ; condición ; incremento/decremento)
{
   grupo_de_sentencias;
}
```

La primera parte de la construcción `for` acostumbra a ser una sentencia de asignación donde se inicializa alguna variable que controla el número de veces que debe ejecutarse el cuerpo del bucle. Esta sentencia se ejecuta una sola ocasión, antes de entrar por primera vez al cuerpo del bucle.

La segunda parte corresponde a la condición que indica cuándo finaliza el bucle, de la misma forma que en las construcciones iterativas anteriores. En este caso, la condición se evalúa antes de ejecutar el cuerpo del bucle, por lo que al igual que en la construcción `while`, el cuerpo puede ejecutarse entre 0 y N veces, donde N depende de la condición.

La tercera parte corresponde normalmente a una sentencia de incremento o decremento sobre la variable de control del bucle. Esta sentencia se ejecuta siempre después de la ejecución del cuerpo del bucle.