CONTENIDOS

1. **La computadora**

2. **Entidades primitivas para el desarrollo de algoritmos.**

3. **Metodología a seguir para la resolución de problemas con computadoras**
   Describir el concepto de algoritmo, diseño de algoritmos como paso previo a la creación de un programa y representación de algoritmos: diagramas de flujo.

4. **Estructuras algorítmicas**
   Estructuras secuenciales, condicionales y cíclicas.

5. **Codificación de algoritmos. Compiladores**
   Tipos de lenguajes, paradigmas de programación, estilos de programación.

1
La computadora

Introducción a la ciencia de la computación y a la programación

Las **computadoras** electrónicas modernas son uno de los productos más importantes del siglo XX.

Son una herramienta esencial en muchas áreas: industria, ciencia, educación, etc

El papel que juegan los lenguajes de programación y los programas es esencial; sin una lista de instrucciones a seguir, las computadoras son inútiles.

Los lenguajes de programación nos permiten escribir esos programas y por consiguiente comunicarnos con las computadoras.

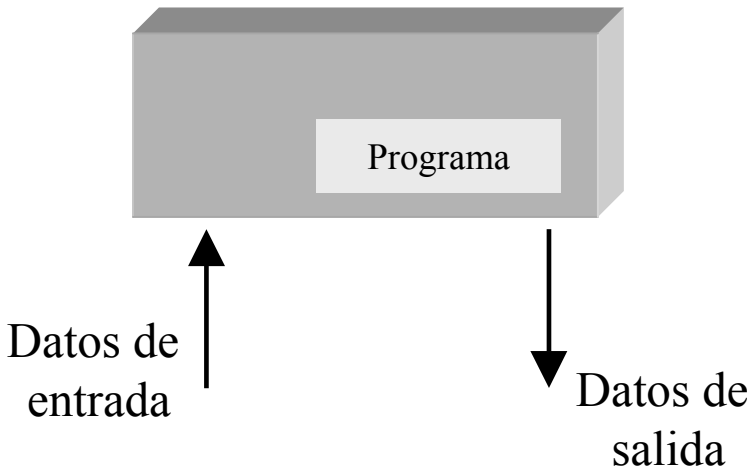
Las computadoras son una herramienta para resolver problemas. La resolución de un problema exige al menos los siguientes pasos:

1. Definición o análisis del problema
2. Diseño del algoritmo
3. Transformación del algoritmo en un programa: codificación
4. Ejecución y validación del programa

LA COMPUTADORA

Una computadora es un dispositivo electrónico utilizado para procesar información y obtener unos resultados.

Computadora



Los datos se pueden introducir en la computadora por la entrada y a continuación se procesan para producir una salida

LA COMPUTADORA

Los datos de entrada y los datos de salida pueden ser cualquier cosa:

texto sonido dibujos

La computadora tiene 2 partes principales:

HARDWARE

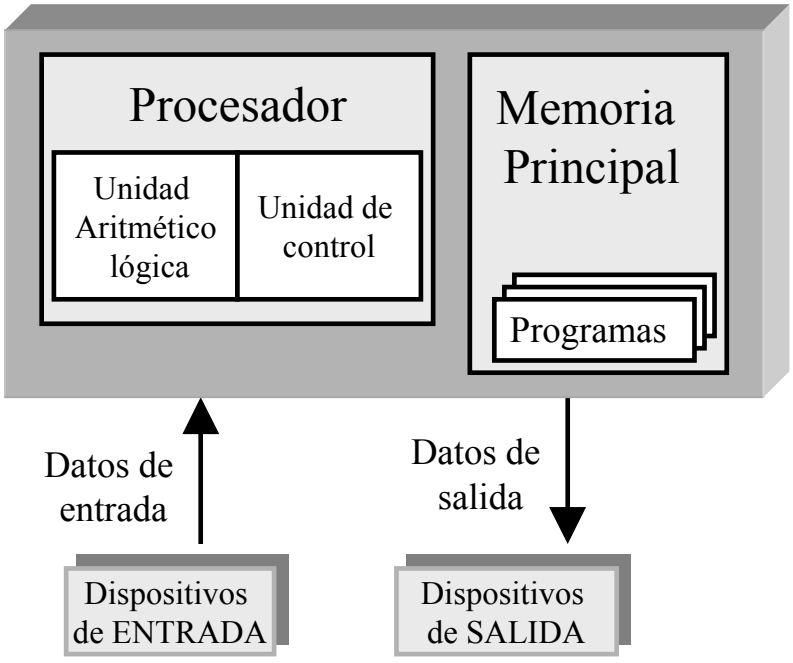
Es la parte **física** de la computadora.
Los componentes físicos que constituyen la computadora son por ejemplo:
ratón, monitor, teclado, ...

SOFTWARE

Es la parte **lógica** de la computadora. El conjunto de instrucciones que hacen funcionar a la computadora se llama **PROGRAMA**. Al conjunto de programas se llama **SOFTWARE** y se encuentran almacenados en la memoria de la computadora.

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

Consta de 2 componentes principales:



ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal

Memoria de acceso aleatorio

- Se utiliza para almacenar información (RAM)

- La información almacenada puede ser de 2 tipos:

Instrucciones de un programa	**Datos** con los que operan las instrucciones

- Para que un programa se pueda ejecutar debe ser situado en la memoria central (carga del programa).

- En la memoria central hay también un espacio de almacenamiento temporal que necesita el programa para poder ejecutarse.

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal. El almacenamiento: El **BIT**

Concepto fundamental para el almacenamiento de datos

Un **bit** es cualquier dispositivo capaz de estar en dos estados:

encendido o apagado,
abierto o cerrado, …

Una tecnología muy utilizada para representar bits en las computadoras es el **condensador.** Los condensadores son dispositivos electrónicos que pueden estar en uno de dos estados:

cargado* o *descargado

Rara vez se utilizan los términos cargado o descargado para referirse al estado de un bit en una máquina, ya que no todos los bits se representan mediante la tecnología de condensadores.

En su lugar son más comunes los términos ***uno*** y ***cero***.

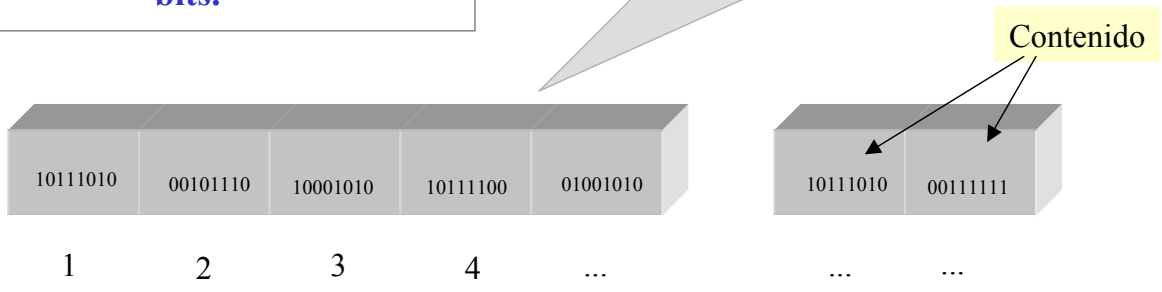
ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal: organización de la memoria.

Los datos que se almacenan en la memoria principal se representan mediante un sistema de codificación de 0's y 1's, de forma que cada combinación de éstos bits se corresponde con un símbolo, como por ejemplo, las letras del alfabeto, los números o los signos de puntuación.

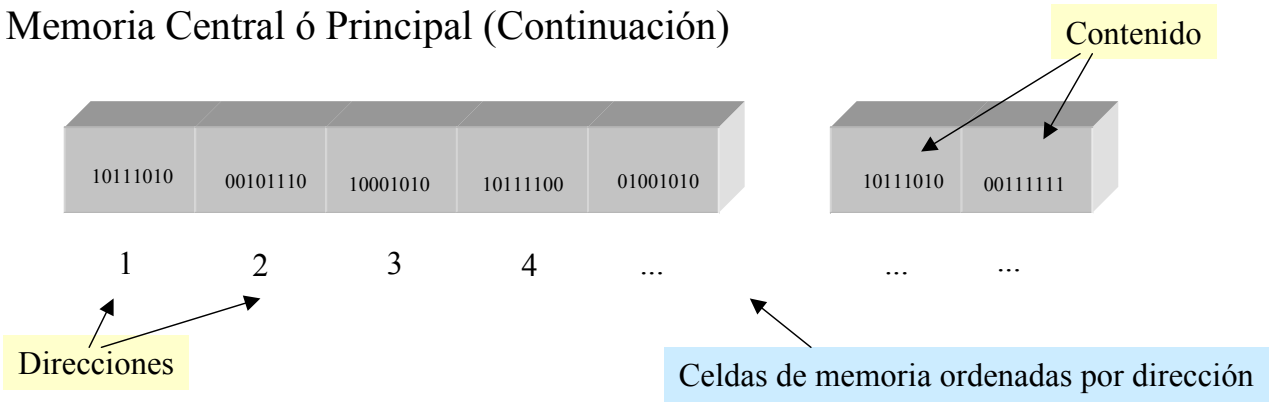
La memoria principal es una vasta colección de circuitos capaz de almacenar un gran número de bits.

Esta colección de bits se divide en unidades más manejables llamadas **CELDAS (8 bits)**



ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal (Continuación)



Existen 2 conceptos importantes asociados a cada celda ó posición de memoria:

- Dirección de la celda: indica la posición en memoria.
- Contenido de la celda: es la información almacenada en dicha posición de memoria (información codificada).

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal. Codificación de la información para su almacenamiento

Un procedimiento para **representar símbolos** consiste en diseñar un código en el que se asigne a los distintos símbolos
(letras, signos de puntuación, …)
una combinación única de bits.

Existe un código estándar llamado **ASCII**.

El código ASCII emplea 7 bits para representar un símbolo y se almacena en una celda de 8 bits (sobra un bit).

Cada celda consta de 8 bits (1 byte)
Un byte tiene capacidad para almacenar un carácter de información

Tabla ASCII

Símbolo	Código ASCII	
0	0110000	48
1	0110001	49
2	0110010	50
3	0110011	51
4		52
5	….	53
6		54
7	0110111	55
8	0111000	56
9	0111001	57

Símbolo	Código ASCII	
A	1000001	65
B	1000010	66
C	1000011	67
D	1000100	68
E	1000101	69
…	…	…
X	1011000	88
Y	1011001	89
Z	1011010	90

Símbolo	Código ASCII	
a	1100001	97
b	1100010	98
c	1100011	99
d	1100100	100
…	…	…
w	1110111	119
x	1111000	120
y	1111001	121
z	1111010	122

$$0 < 1 < 2 \ldots . < 9 < A < B < \ldots < Z < a < b < \ldots < z$$

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal (Continuación)

  Ejemplo:

  Si deseamos almacenar la frase: *"Hola mundo"*

  la computadora utilizará 10 bytes consecutivos de memoria.

Este método de almacenar información resulta ineficiente cuando los datos que se desean almacenar son numéricos.

La **representación de valores numéricos** se realiza de forma diferente, ya que para almacenar el número 3000, necesito 4 celdas de memoria (32 bits).

Una estrategia más eficiente consiste en almacenar el valor en su representación en base dos (representación binaria).

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- Memoria Central ó Principal: **representación binaria**

Código binario	Número
00000000	0
00000001	1
00000010	2
00000011	3
00000100	4
…	
11111111	255

En **un byte** podemos almacenar cualquier entero entre 0 y 255

Si contamos con **dos bytes**, podemos almacenar enteros desde el 0 hasta el 65535

Esto representa una mejoría notable respecto a la otra forma de almacenamiento.

ORGANIZACIÓN FÍSICA DE UNA COMPUTADORA

- CPU (Unidad Central de Proceso) ó procesador
  - Procesa y manipula la información almacenada en memoria
    * Recupera la información desde memoria (datos y programas)
    * Controla y realiza operaciones con los datos
    * Almacena resultados de los procesos en dicha memoria para su utilización posterior.

  - Consta de dos componentes:

Unidad de control

Coordina las actividades de la computadora

Qué operaciones se deben realizar y en qué orden

Unidad Aritmético-lógica

Dispositivo físico capaz de realizar operaciones aritméticas y lógicas
(suma, resta, multiplicación, …)

2

Entidades primitivas para el desarrollo de algoritmos

CONSTANTES Y VARIABLES

- Son *porciones de memoria que almacenan un valor.*
- Una **constante** es un dato cuyo valor no cambia durante la ejecución del programa.

  Ejemplo: pi = 3,1416
- Las **variables** son palabras que manipulan datos. Dicho valor puede ser modificado en cualquier momento durante la ejecución del programa.
- Tanto las variables como las constantes están constituidas por un **nombre** y un **valor.** El nombre lo llamaremos **identificador.**
- El valor de la variable puede ser simple o compuesto. Dependiendo del valor de la variable, decimos que dicha variable es de un tipo de dato. Por ejemplo, si el valor de la variable es un entero, decimos que la variable es de tipo entero.

IDENTIFICADORES

- Los identificadores representan los datos de un programa.
- Es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, y nos permite acceder al contenido de dicha posición.
- Reglas para formar un identificador:

  - Debe comenzar con una letra.

  - Después de la primera letra pueden aparecer otras letras, dígitos y caracteres.

  - No debe contener espacios en blanco.

Ejemplos:

SUMA	n_primo
NUMERO1	num_horas

TIPOS DE DATOS SIMPLES

- Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del valor que puede tomar una variable.

Datos Numéricos: son los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.

Datos Lógicos: Son aquellos que sólo pueden tener dos valores (cierto o falso). Son el resultado de una comparación entre otros datos (numéricos o alfanuméricos).

Datos Alfanuméricos: Es una secuencia de caracteres, por ejemplo, nombres de personas, direcciones, etc. Este tipo de datos se representan encerrados entre comillas. Ejemplos: "Hola amigos" ó "1997"

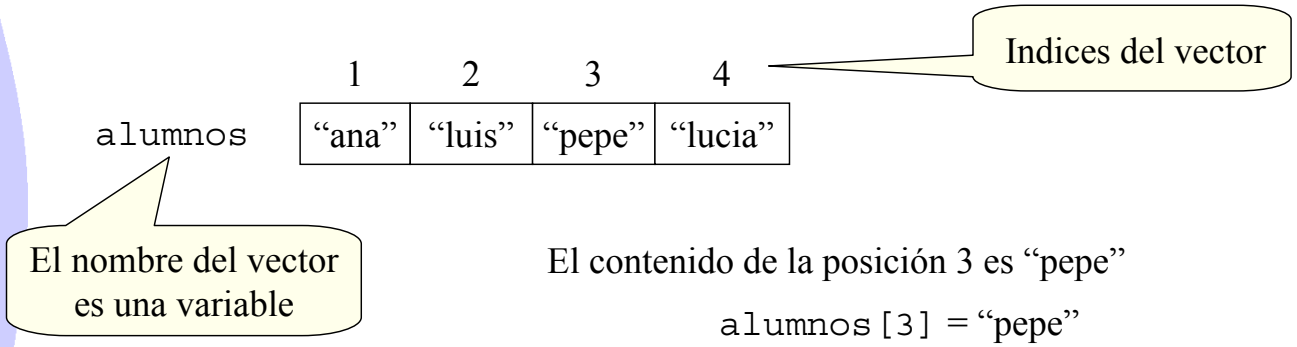
TIPOS DE DATOS ESTRUCTURADOS: ARRAYS

- Son un conjunto finito de valores
- Un *Array* es una estructura de datos que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo. Los arrays se caracterizan por:
  - Almacenan los elementos en posiciones contiguas de memoria.
  - Tienen un mismo nombre de variable que representa a todos los elementos.
  - Para hacer referencia a esos elementos es necesario utilizar un índice que especifica el lugar que ocupa cada elemento dentro del array.
- Tipo de Arrays : Vectores y Matrices.

VECTORES

- Es un array de "N" elementos donde "N" recibe el nombre de *longitud* o tamaño del vector.
- Para hacer referencia a un elemento del vector se usa el nombre del mismo, seguido del *índice* (entre corchetes), el cual indica una posición en particular del vector.

Esto es un vector de 4 elementos



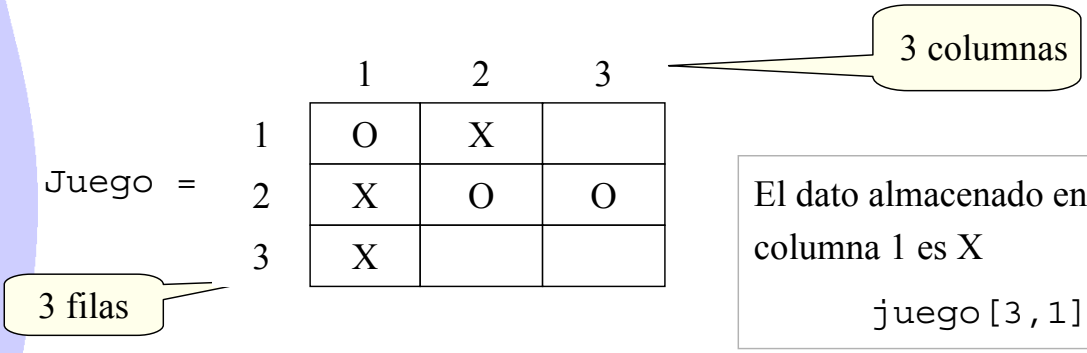
El contenido de la posición 3 es "pepe"

`alumnos[3]` = "pepe"

MATRICES

- Es un array de M * N elementos organizados en dos dimensiones donde "M" es el numero de filas y "N" el numero de columnas.
- Para hacer referencia a un elemento de la matriz, se usa el nombre seguido de dos *índices* (entre corchetes), que indican la fila y la columna donde se encuentra almacenado el dato.

Esto es una matriz 3 * 3 elementos



El dato almacenado en la fila 3 y columna 1 es X

`juego[3,1]` = X

EXPRESIONES

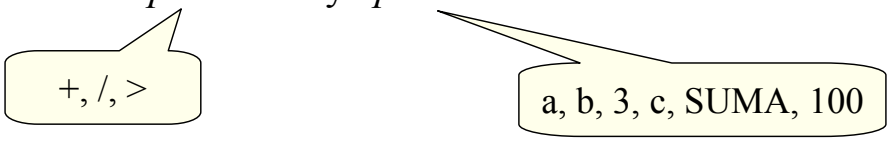
- Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

  Ejemplos:

  a + (b + 3) / c

  SUMA > 100

- Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.
- Una expresión consta de *operadores* y *operandos*.



3

Metodología a seguir para la resolución de problemas con computadoras

RESOLUCIÓN DE PROBLEMAS

Concepto de ALGORITMO

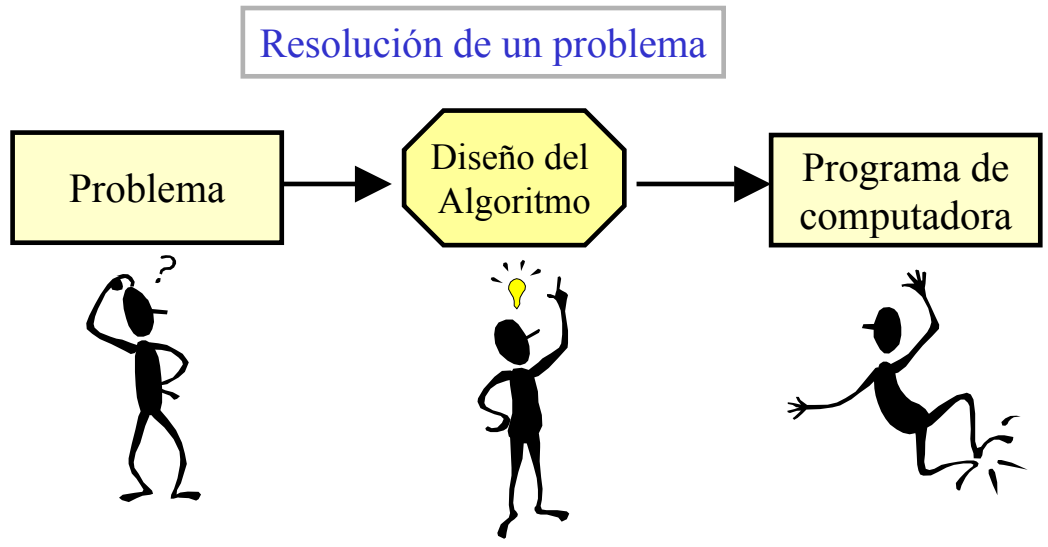
El objetivo fundamental de éste curso es enseñar a resolver problemas mediante una computadora. Para ello necesitamos estudiar una metodología que nos ayudará en dicha tarea.

El eje central de dicha metodología es el concepto de algoritmo.

- Un algoritmo es una serie de pasos ordenados que describen el proceso que se debe seguir para dar solución a un problema específico.
- Un algoritmo es un conjunto finito de instrucciones que especifican la secuencia de operaciones a realizar en orden para resolver un problema.

ALGORITMOS

Para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático. Esto significa que sólo se puede llegar a realizar un buen programa con el diseño previo de un algoritmo.



ALGORITMOS

- Sin algoritmo no puede existir programa.
- Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora donde se ejecutan.
- Un lenguaje de programación es un medio para expresar un algoritmo y una computadora es un procesador para ejecutarlo.

La definición de un algoritmo debe describir tres partes:

1. Entrada: El algoritmo tiene 0 ó más entradas. Son datos que se entregan al algoritmo antes de su ejecución.

2. Proceso: Pasos del algoritmo

3. Salida: El algoritmo tiene 1 ó más salidas. Es el resultado de aplicar los pasos del algoritmo.

ALGORITMO

Características de los algoritmos.

Las características fundamentales son:

- **Debe ser preciso**

  Debe especificar las acciones que se van a realizar y el orden de dichas acciones.

- **Debe estar definido**

  Si se sigue el algoritmo dos veces, se debe obtener el mismo resultado cada vez.

- **Debe ser finito**

  Debe tener un número finito de pasos.

ALGORITMO

Ejemplo:

Realizar la suma de todos los números pares entre 2 y 100.

El problema consiste en sumar 2+4+6+ …+ 98+100.

Utilizamos las palabras SUMA y NUMERO para representar las sumas sucesivas (2+4), (2+4+6), etc.

El algoritmo es el siguiente:

SUMA Y NUMERO son variables

1. Inicialmente SUMA vale 0
2. Inicialmente el valor de NUMERO es 2
3. Sumar NUMERO a SUMA
4. Aumentar el valor de NUMERO en dos unidades
5. Si el valor de NUMERO ≤100 entonces ir al paso 3
6. En caso contrario, finalizar el proceso.

RESOLUCIÓN DE PROBLEMAS

Las fases de resolución de un problema:

1. Definición o análisis del problema
   Consiste en entender el problema que se quiere resolver.
   En ésta fase se determina *QUÉ* debe hacer el algoritmo.

2. Diseño del algoritmo
   Pensar e idear un plan para resolver el problema. En ésta fase de determina *CÓMO* se hace la tarea solicitada.

3. Transformación del algoritmo en un programa: codificación
   Llevar a cabo el plan, es decir, representar el algoritmo en forma de programa.

4. Ejecución y validación del programa
   Evaluar el programa en cuanto a capacidad para resolver el problema planteado.

Definición o análisis del problema

Para poder definir correctamente un problema, es conveniente responder a las siguientes preguntas:

- ¿Qué entradas se requieren?
- ¿Cuál es la salida deseada?
- ¿Qué método produce la salida deseada?

Ejemplo:

Se desea obtener el coste final de un automóvil sabiendo que el importe inicial es de 12.000 € y los descuentos a aplicar son del 10% si la venta se produce antes de Junio de 2004 y del 20% si se produce con posterioridad.

Entradas: Coste original y los descuentos según el mes.
Salidas: Coste del automóvil hasta Junio, coste del automóvil a partir de Junio.
Proceso: Cálculo del descuento aplicado hasta Junio y a partir de Junio.

Diseño del algoritmo

Para diseñar un algoritmo, es decir, para especificar el **CÓMO** se hace una determinada tarea, se dispone de dos herramientas:

- ★ **Diagramas de flujo**
- ★ **Pseudocódigo**

Diagramas de flujo

⇒ Es la representación gráfica de un algoritmo.

⇒ Permite representar la secuencia de operaciones que se deben realizar para la resolución de un problema, es decir, permite representar el flujo de información desde su entrada hasta su salida.

⇒ Dispone de un conjunto de **símbolos gráficos** con significado referente al tipo de instrucción que se va a realizar.

⇒ Dichos símbolos van unidos con **flechas** que indican el orden de secuencia a seguir.

Diseño del algoritmo

Para diseñar un algoritmo, es decir, para especificar el **CÓMO** se hace una determinada tarea, se dispone de varias herramientas:

- ★ **Diagramas de flujo**
- ★ **Pseudocódigo**

Pseudocódigo

⇒ Es un lenguaje utilizado para definir algoritmos con una sintaxis muy parecida a la de un lenguaje de programación.

⇒ Las instrucciones se escriben en palabras similares al inglés o al español, facilitando así la comprensión el algoritmo.

⇒ La ventaja es que es muy fácil pasar de pseudocódigo a un lenguaje de programación.

Diseño del algoritmo: Diagramas de flujo

Los símbolos gráficos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI).

Los más frecuentemente utilizados son:

	Indica el **inicio** y **final** del diagrama de flujo
	Indica la entrada y salida de **datos**
	Símbolo de **proceso**. Indica la realización de una operación.
	Símbolo de **decisión.** Indica operaciones de comparación entre datos. En función del resultado se sigue por distintos caminos.
	Llamada a otro proceso complejo. Llamada a subrutina.

Diseño del algoritmo: Diagramas de flujo

Otros símbolos:

	Indica la salida de información por impresora.
	Conector. Representa la continuidad del diagrama.
	Línea de flujo. Indica el sentido de ejecución de las operaciones

Recomendaciones para el diseño de Diagramas de Flujo

- Se deben usar solamente líneas de flujo horizontales y/o verticales.
- Se debe evitar cruce de líneas utilizando los conectores.

Diseño del algoritmo: Diagramas de flujo

Ejemplo:

Realizar la suma de todos los números pares entre 2 y 100.

Descripción del algoritmo

1. Inicialmente SUMA vale 0.

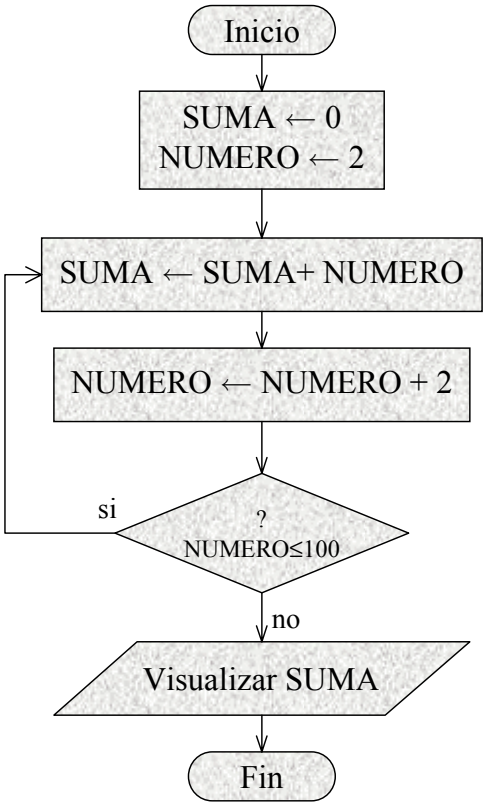
2. Inicialmente el valor de NUMERO es 2.

3. Sumar NUMERO a SUMA.

4. Aumentar el valor de NUMERO en dos unidades.

5. Si el valor de NUMERO≤ 100 entonces ir al paso 3.

6. En caso contrario, devolver el valor de SUMA y finalizar el proceso.



Diseño del algoritmo: Diagramas de flujo

Ejemplo:

Realizar la suma de todos los números pares entre 2 y 100.

Pseudocódigo

1. SUMA = 0

2. NUMERO = 2.

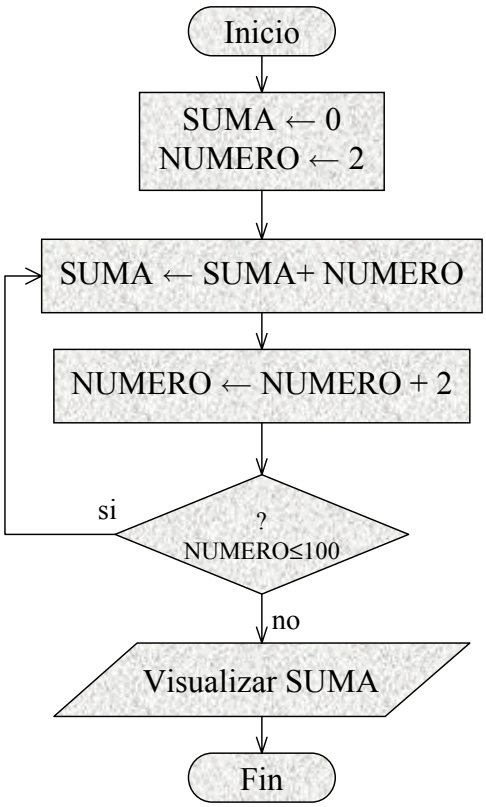
3. mientras NUMERO ≤ 100

SUMA = SUMA + NUMERO

NUMERO = NUMERO + 2

fin_mientras

4. visualizar SUMA y finalizar el proceso.



EJERCICIOS

Ejercicio 1:

Escribir un algoritmo que permita introducir números por teclado hasta que la suma de todos ellos sea mayor que 30.

Ejercicio 2:

Escribir un algoritmo que escriba en pantalla el número de elementos positivos, negativos y ceros que contiene una tabla de 500 posiciones de contenido numérico.

Ejercicio 3:

Supongamos que se proporciona una secuencia de números y se desea contar e imprimir el número de ceros de la secuencia.

Ejercicio 4:

Escribir un algoritmo que calcule el área de un triángulo en función de la base y la altura.

EJERCICIOS

Ejercicio 5:

Dados 3 números, determinar si la suma de cualquiera de ellos es igual al tercer número. Si se cumple esta condición, escribir "IGUALES" y, en caso contrario, escribir "DISTINTAS".

Ejercicio 6:

Escribir un algoritmo que lea 3 números y a continuación escriba el mayor de los 3.

Ejercicio 7:

Diseñar un algoritmo que le un número positivo y escriba por pantalla la palabra "Hola" las veces que indique dicho número.

Ejercicio 8:

Dada una palabra de 10 caracteres, averiguar si es palíndroma.(Utilizar el tipo de datos array).

EJERCICIOS

Ejercicio 9:

Diseñar un algoritmo que lea un número positivo y escriba por pantalla todos los números positivos menores que él en orden descendente.
(Si se lee el número 6, entonces se escribe 5 4 3 2 1)

Ejercicio 10:

*Escribir un algoritmo que visualice una tabla de ceros de tamaño M*N, donde M y N son datos leídos por pantalla.*
Por ejemplo si M = 3 y N=4, se visualiza la siguiente tabla:

0 0 0 0
0 0 0 0
0 0 0 0

Ejercicio 11:

Diseñar un algoritmo que lea un número positivo y averigüe si es primo.

EJERCICIOS

Ejercicio 12:

Diseñar un algoritmo para visualizar el siguiente dibujo:

```
*
***
*****
*******
*********
*******
*****
***
*
```


EJERCICIOS

Ejercicio 13:

Dada una matriz de enteros de tamaño 5×5 , diseñar un algoritmo que calcule la suma de todas las componentes.

Por ejemplo, si la matriz es:

1	2	3	2	1
1	1	1	1	1
2	2	2	2	2
0	0	1	0	1
2	1	3	1	2

el algoritmo debe devolver el valor 35.

4
Estructuras algorítmicas

CLASIFICACIÓN

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten realizar ciertos procesos específicos que nos lleven a la solución de problemas.

Estas estructuras se clasifican de acuerdo a su complejidad:

- **Estructuras Algorítmicas**
  - **Secuenciales**
    - **Entrada**
    - **Salida**
    - **Asignación**
  - **Condicionales**
    - **Simples**
    - **Múltiples**
  - **Cíclicas**
    - **Hacer para**
    - **Hacer mientras**
    - **Repetir hasta**

La estructura secuencial es aquella en la que una acción o instrucción sigue a otra en un orden secuencial.

Representación en pseudocódigo

```
Inicio
Acción 1
Acción 2
  ..
  ..
Acción n
Fin
```

Lectura o Entrada

Consiste en recibir desde un dispositivo de entrada, un valor.

```
leer DATO
```

DATO es la variable que recibe el valor

Escritura

Consiste en mandar por un dispositivo de salida un resultado o mensaje.

```
escribir "La suma de los números es: " SUMA
```

En la pantalla aparece el texto: La suma de los números es:

SUMA es la variable que contiene un valor

Asignación

Consiste en el paso de valores o resultados a una zona de memoria.

Simples

```
a = 15
```

Consiste en pasar un valor constante a una variable

Contador

```
a = a + 1
```

Consiste en usarla como un verificador del numero de veces que se realiza un proceso

Acumulador

```
suma = suma + 4
```

Consiste en usarla como un sumador en un proceso

De trabajo

```
a = c +b / 4
```

Puede recibir el resultado de una operación matemática que involucre muchas variables

Estructuras secuenciales

Ejemplo:

Se desea conocer el porcentaje de hombres y el porcentaje de mujeres que realizan un Máster al finalizar su carrera universitaria. Se supone que los datos Nº DE HOMBRES y Nº De MUJERES se leen desde un dispositivo de entrada.

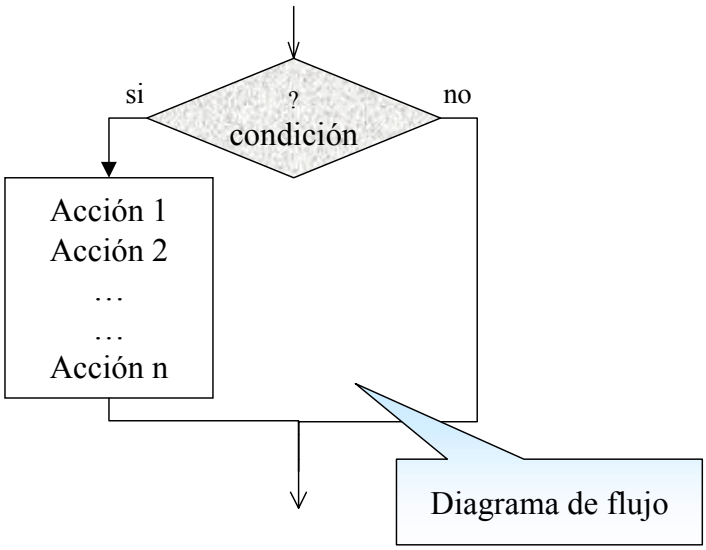
Estructuras condicionales

Las estructuras condicionales comparan una variable con otro(s) valor(es), para que en base al resultado de esta comparación, se siga un camino dentro del programa.

La comparación se puede hacer contra otra variable o contra una constante, según se necesite.

Simples

```
Si <condición> entonces
    Acción 1
    Acción 2
      ..
      ..
    Acción n
Fin_Si
```



Dobles

```
Si <condición> entonces
    Acción 1
    Acción 2
      ..
      ..
    Acción n
Si no
    Acción 1'
    Acción 2'
      ..
      ..
    Acción n'
Fin_Si
```

Acciones a realizar si se cumple la condición

Acciones a realizar en el caso de que **NO** se cumpla la condición

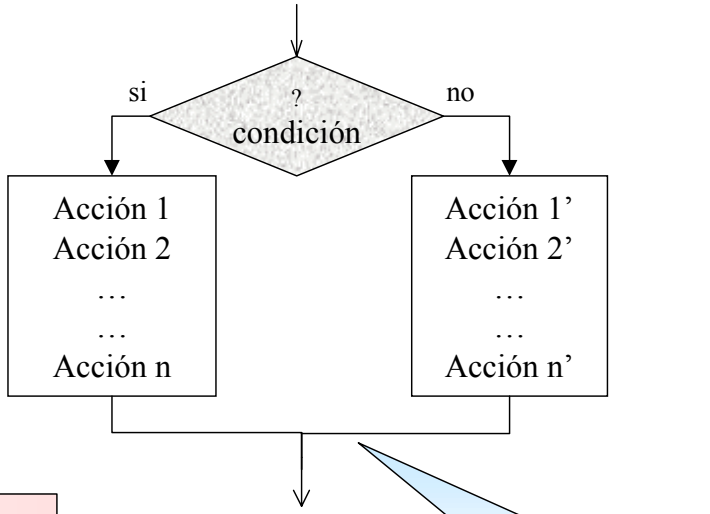


Diagrama de flujo

Múltiples

Son tomas de decisión que permiten comparar una variable con distintos valores posibles, ejecutando para cada caso una serie de instrucciones específicas.

```
Si <condición> entonces
    Acción(es)
Si no
    Si <condición> entonces
        Acción(es)
    Si no
        Acción(es)
    Fin_Si
Fin_Si
```

```
Caso  DATO
   op1:
      Acción(es)
   op2:
      Acción(es)

    . . .
   opN:
      Acción(es)
Fin_caso
```

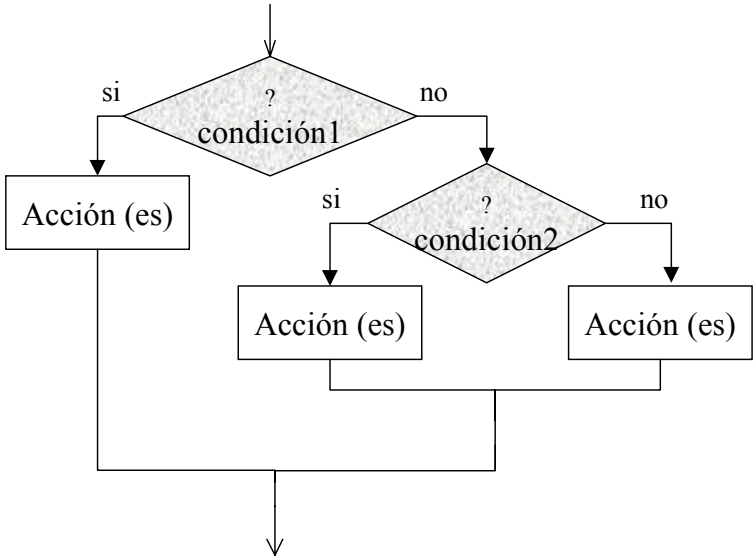
DATO es la variable que se compara

Si **DATO** es igual a **op1**, se ejecutan las acciones correspondientes

Múltiples

Su representación en diagrama de flujo se puede hacer de varias formas:

```
Si <condición1> entonces
    Acción(es)
Si no
    Si <condición2> entonces
        Acción(es)
    Si no
        Acción(es)
    Fin_Si
Fin_Si
```

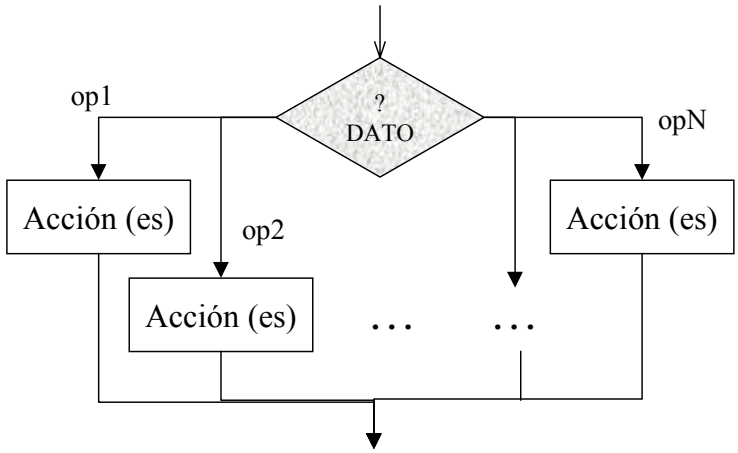


Múltiples

Diagrama de flujo para la representación de **CASOS**:

```
Caso  DATO
   op1:
      Acción(es)
   op2:
      Acción(es)

    . . .
   opN:
      Acción(es)
Fin_caso
```



Estructuras condicionales

Ejemplo 1:

En un almacén se hace un 20% de descuento a los clientes cuya compra supere los 1000 €. ¿Cual será el cantidad que pagará un cliente por su compra? Suponemos que el importe de la compra es un dato que se lee desde un dispositivo de entrada.

Ejemplo 2:

Escribir un algoritmo que lea 3 números y a continuación escriba el mayor de los 3.

Ejemplo 3:

Escribir un algoritmo que lea un número comprendido entre el 1 y el 12, y que imprima por pantalla el mes al que corresponde dicho número.

Estructuras cíclicas

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad especifica de veces.

Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).

Los ciclos se clasifican en:

Hacer para — Ciclos con un número DETERMINADO de iteraciones

Hacer mientras
Repetir hasta — Ciclos con un número INDETERMINADO de iteraciones

Hacer para

Ciclos con un Numero **Determinado** de Iteraciones.

Son aquellos en que el numero de iteraciones se conoce antes de ejecutarse el ciclo.

```
Hacer Para i = inf hasta i = sup
      Acción 1
      Acción 2
       . . .
      Acción N
Fin_Para
```

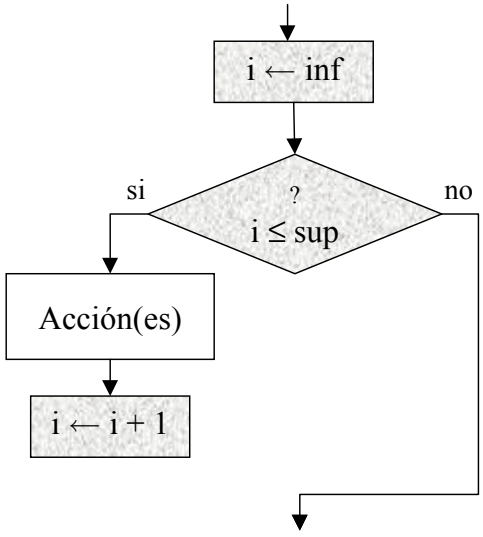
inf y **sup** representan el límite inferior y el límite superior del bucle.

La variable **i** es la variable de control del ciclo. El ciclo se repite desde el límite inferior, hasta que la variable de control llegue la límite superior

Hacer para

Son aquellos en que el numero de iteraciones se conoce antes de ejecutarse el ciclo.

```
Hacer Para i = inf hasta i = sup
      Acción 1
      Acción 2
       . . .
      Acción N
Fin_Para
```

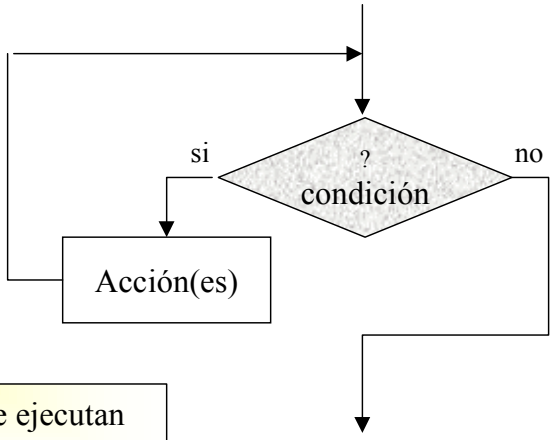


Hacer Mientras

Ciclos con un Numero **Indeterminado** de Iteraciones.

Son aquellos en que el numero de iteraciones no se conoce con exactitud, ya que está dado en función de un dato dentro del programa

```
Hacer
Mientras <condición>
    Acción 1
    Acción 2
     . . .
    Acción N
Fin_Mientras
```



El grupo de acciones se ejecutan mientras se cumpla la condición

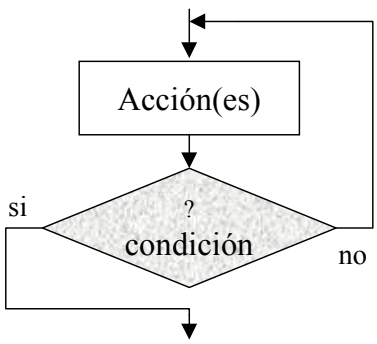
Repetir Hasta

Esta es una estructura similar a la anterior

Repite un proceso una cantidad de veces, pero a diferencia de la anterior, lo hace ***hasta que la condición se cumple*** y no mientras, como en el Hacer-Mientras.

Por otra parte, esta estructura **permite realizar el proceso al menos una vez**, ya que la condición se evalúa al final del proceso, mientras que en el Hacer-Mientras puede ser que nunca llegue a entrar si la condición no se cumple desde un principio.

```
Repetir
      Acción 1
      Acción 2
       . . .
      Acción N
Hasta  <condición>
```



Ejemplo 1:

La calificación final de un alumno en la asignatura de cálculo se obtiene como media aritmética de 7 notas. Calcular dicha calificación suponiendo que las 7 notas se leen desde un dispositivo de entrada.

Ejemplo 2:

Encontrar el mayor valor de un conjunto de números leídos desde un dispositivo de entrada.

5
Codificación de algoritmos

CODIFICACIÓN: LOS LENGUAJES DE PROGRAMACIÓN

Un lenguaje de programación es un medio para expresar un algoritmo y una computadora es un procesador para ejecutarlo.

Una vez que tenemos diseñado un algoritmo, el procesador debe ser capaz de entender su significado, es decir, comprender las instrucciones de cada paso y realizar las operaciones correspondientes.

Por tanto, el algoritmo debe expresarse en un formato denominado **PROGRAMA** y éstos programas se escriben mediante **LENGUAJES DE PROGRAMACIÓN**.

Los principales niveles de lenguajes son 3:

- Lenguaje máquina
- Lenguaje de bajo nivel
- Lenguajes de alto nivel

CODIFICACIÓN: LOS LENGUAJES DE PROGRAMACIÓN

Instrucciones a la computadora

Los diferentes pasos de un algoritmo, se expresan mediante instrucciones, por tanto, un programa no es más que una secuencia de instrucciones, cada una de las cuales especifica ciertas operaciones que tiene que realizar la computadora.

Las instrucciones básicas y comunes a casi todos los programas las podemos agrupar en 4:

Instrucciones de Entrada/Salida
Transferencia de datos entre los dispositivos y la memoria central

Instrucciones aritmético-lógicas
Instrucciones que ejecutan operaciones aritméticas (suma, resta, …) y lógicas (or, and, …)

Instrucciones Selectivas
Permiten la selección de tareas alternativas en función del resultado de expresiones condicionales

Instrucciones repetitivas
Permiten que unas tareas se repitan un número de veces

LENGUAJE MÁQUINA

Los lenguajes máquina son aquellos que son directamente inteligibles por la computadora ya que las instrucciones se expresan con cadenas binarias (0´s y 1's).

Estas instrucciones se llaman **CÓDIGO MÁQUINA** o **CÓDIGO BINARIO**.

VENTAJA

La velocidad de ejecución de los programas es superior a cualquier otro lenguaje.

INCONVENIENTES

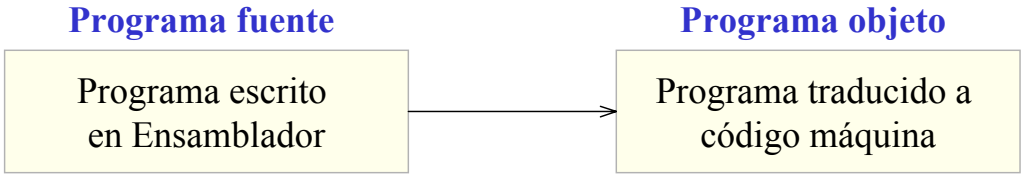
- Son difíciles y lentos de codificar
- Difíciles de mantener y comprender
- Dependen de la CPU donde se ejecutan
- Son poco fiables

LENGUAJE DE BAJO NIVEL

Son un poco más fiables de utilizar que los lenguajes máquina pero también dependen de la CPU donde se ejecutan.

El más importante es el **ENSAMBLADOR**.

Un programa escrito en lenguaje Ensamblador no puede ser ejecutado directamente por la máquina sino que requiere una fase de traducción a lenguaje máquina.



VENTAJA

Más fáciles de codificar que el lenguaje máquina

INCONVENIENTES

- Dependen de la CPU donde se ejecutan
- Son difíciles y lentos de codificar

LENGUAJES DE ALTO NIVEL

Son los mas utilizados por los programadores

Están diseñados para que el escribir y entender los programas sea más fácil que codificar en ensamblador o en código máquina.

VENTAJAS

- El tiempo de formación de los programadores es corto.
- El mantenimiento de los programas y las modificaciones son fáciles de realizar
- No dependen de la CPU donde se ejecutan
- Programas fáciles de entender

INCONVENIENTES

- Ocupan más espacio en memoria
- El tiempo de ejecución es mayor

LENGUAJES DE PROGRAMACIÓN: EJEMPLO

Instrucción típica de suma de dos números:

```
0110   1001   1010   1011
```

Lenguaje máquina

```
ADD M, N, P
```

Lenguaje Ensamblador

```
P = M + N ;
```

Lenguaje de alto nivel

LENGUAJES DE ALTO NIVEL

Al igual que ocurría con el lenguaje ensamblador, los programas escritos mediante un lenguaje de alto nivel deben ser traducidos en programas objeto. Estos traductores se llaman **COMPILADORES o INTERPRETES.**

Estilos de programación

Es importante tener claro las características o estilos de programación para lograr determinar cual es la herramienta o lenguaje ideal según las características del sistema a implementar.

Existen 5 estilos de programación fuertemente conocidos

ESTILOS DE PROGRAMACIÓN

1. Programación Imperativa / Secuencial / Estructurada

Se dispone de un conjunto de operaciones primitivas con una ejecución secuencial. Para programar es necesario diseñar una secuencia adecuada de instrucciones.

Ejemplos: PASCAL, ADA y C

2. Programación Orientada al Objeto (POO)

La programación orientada al objeto esta basada en los objetos, clase, método, envió y recepción de mensajes, herencia y polimorfismo.

Ejemplos: C++, JAVA.

3. Programación orientada al evento

Esta programación es el resultado de la programación orientada al objeto. En este tipo de programación permite trabajar con objetos y clases standard previamente definidas por la aplicación.

Ejemplo: Visual Basic, Delphi y Power Builder

ESTILOS DE PROGRAMACIÓN

4. Programación Funcional

El Programa es una Función (o un grupo de funciones). Una Función puede llamar a otra Función, o el resultado de una Función puede ser usado como el argumento de otra Función.

Ejemplo: Función factorial en ML

*fun Factorial loop(n,f)= if n>0 then factorial loop(n-1,f*n) else f*

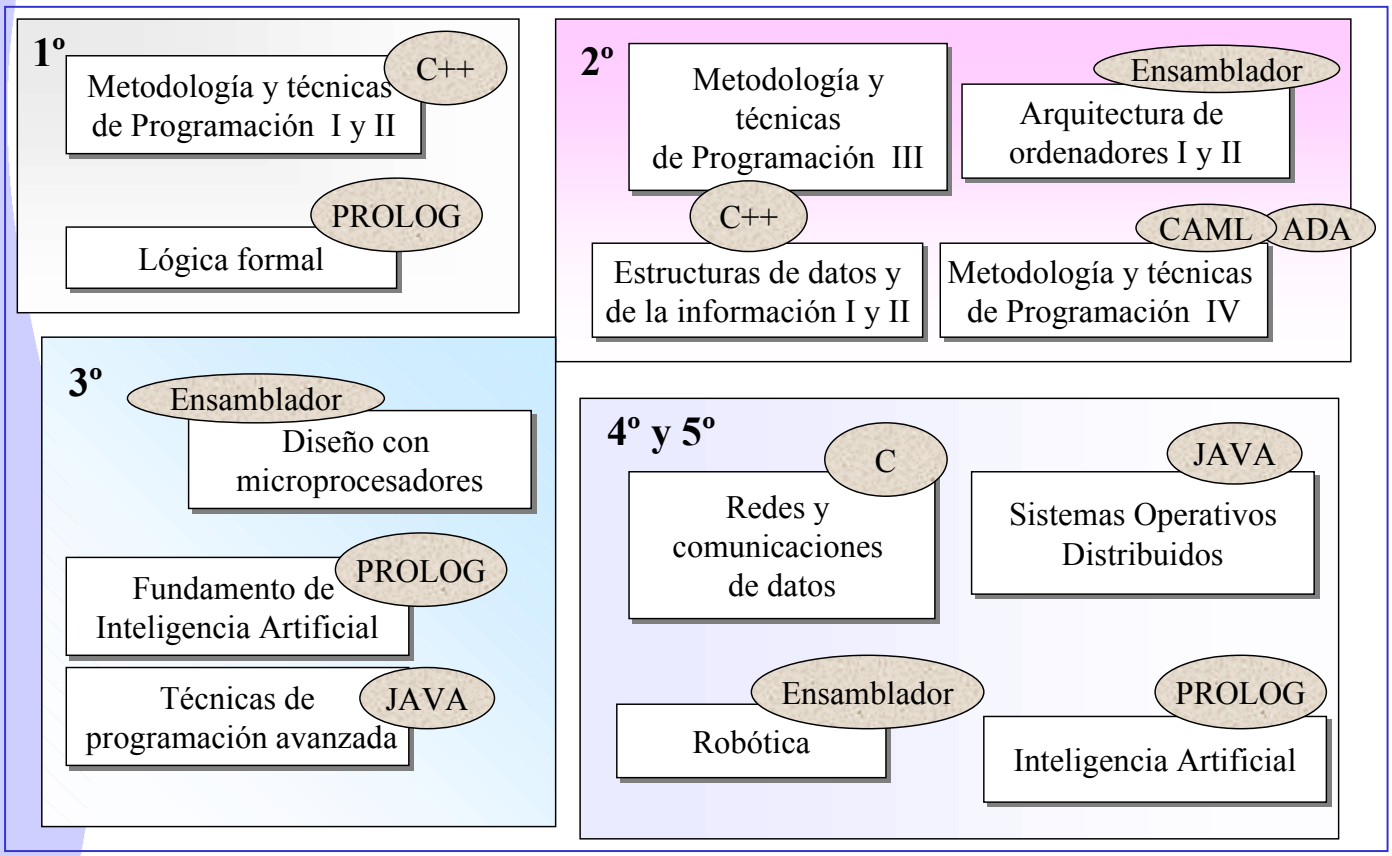
Ejemplos: Haskel, CAML.

5. Programación lógica

La programación Lógica está basada en la noción de relación, debido a que en la relación es un concepto más general de una aplicación. Los lenguajes de Programación Lógica se utilizan en el campo de la Inteligencia Artificial.

Ejemplo: PROLOG

ESTILOS DE PROGRAMACIÓN



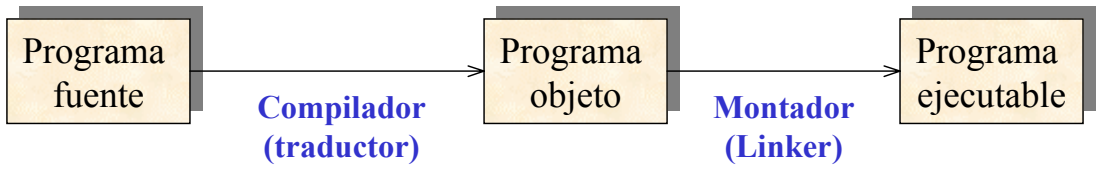
TRADUCTORES: Compiladores e Intérpretes

Sirven para escribir programas que permitan la comunicación entre el usuario y la computadora.

Un ***intérprete*** toma el programa fuente, lo traduce y a continuación lo ejecuta. Un ejemplo de lenguaje interpretado es *Java*.

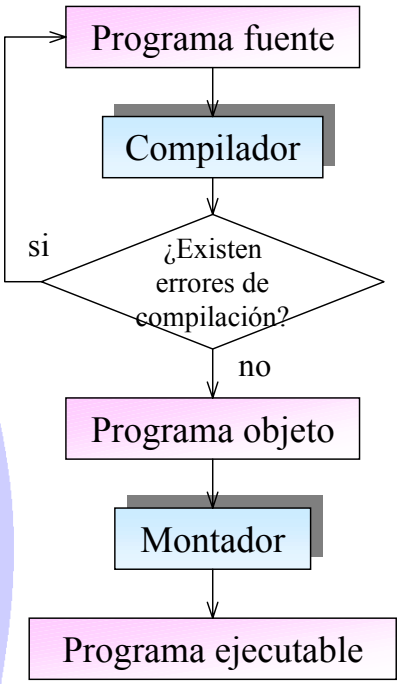
Un ***compilador*** es un programa que se encargan de convertir las instrucciones escritas en un lenguaje de programación en instrucciones escritas en lenguaje máquina (0´s y 1´s) que la computadora pueda entender. Ejemplos de lenguajes compilados son *Pascal, C, C++*.

Fases de la compilación general



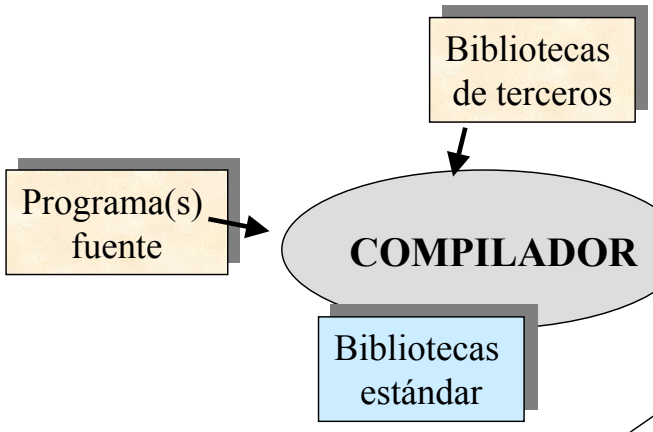
Compilación de programas

Fases de ejecución de un programa

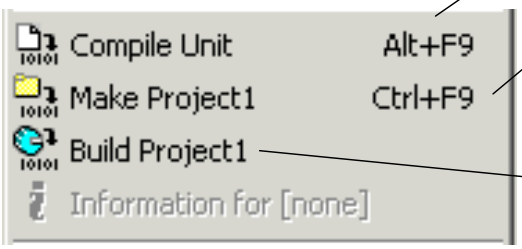


1. Escritura del programa fuente mediante un editor.
2. Traducir el programa mediante un compilador.
3. Verificar y corregir los errores de compilación.
4. Obtención del programa objeto.
5. Obtener el programa ejecutable mediante el montador.
6. Se ejecuta el programa y si no existen errores, se tendrá una salida.

Fases de la compilación



Opciones en C++



Compila cualquier código fuente que ha sido modificado desde la última compilación.

Genera el archivo ejecutable, compilando y enlazando los archivos modificados o que no existan.

Genera el archivo ejecutable, compilando y enlazando todos los archivos.