Pseudocódigo

- El **pseudocódigo** es una **descripción de alto nivel de un algoritmo** que emplea una mezcla de **lenguaje natural** con algunas **convenciones sintácticas** propias de lenguajes de programación, a usar (es un supuesto lenguaje) .

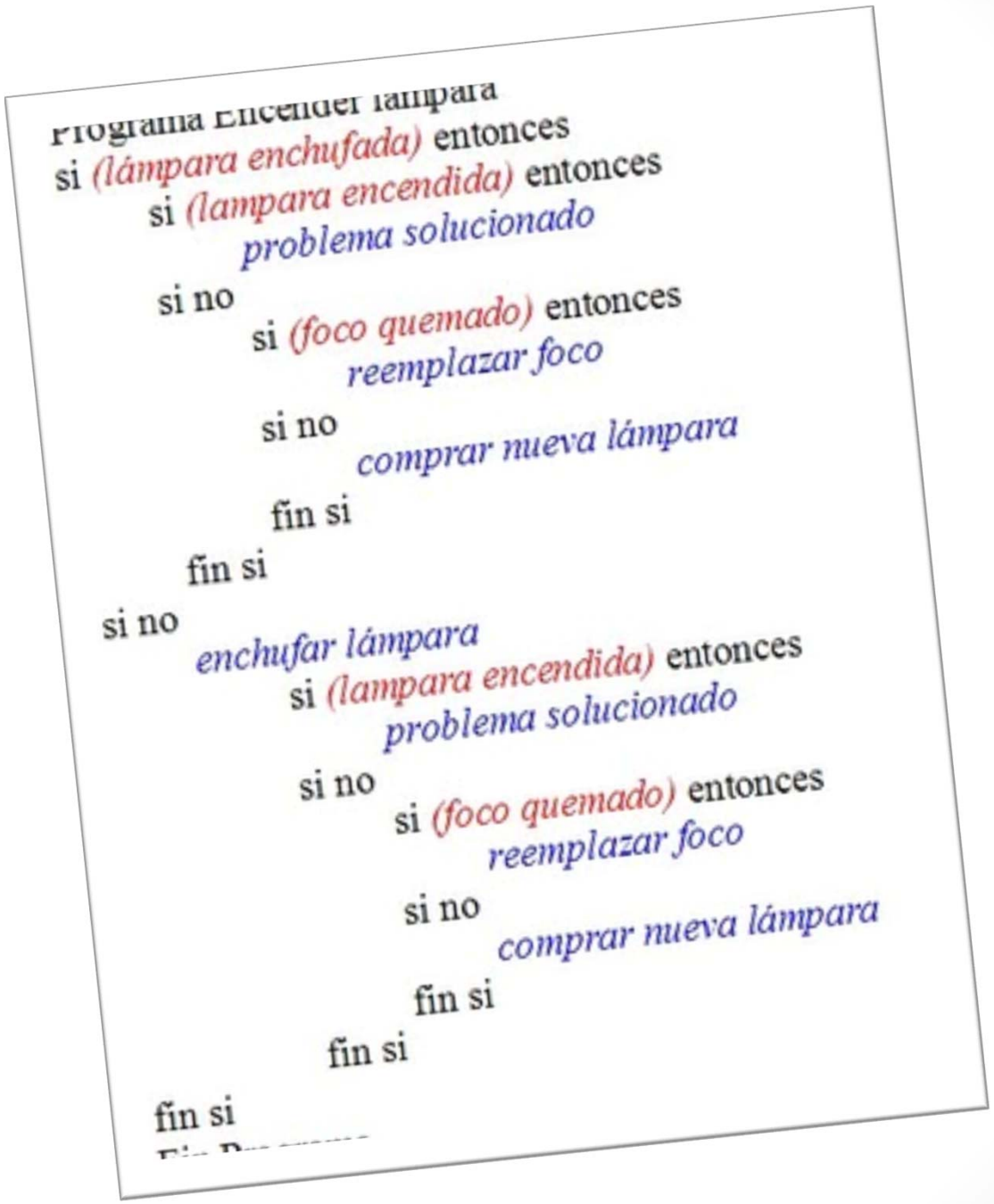
- Es utilizado para describir algoritmos de manera formal en libros y publicaciones científicas, y como producto intermedio durante el desarrollo de un algoritmo.

- El **pseudocódigo** está pensado para **facilitar a las personas el entendimiento de un algoritmo**, y por lo tanto puede omitir detalles irrelevantes que son necesarios en una implementación.

- Programadores diferentes suelen utilizar **convenciones distintas**, que pueden estar basadas en la sintaxis de lenguajes de programación concretos. Sin embargo, **el pseudocódigo en general es comprensible sin necesidad de conocer o utilizar un entorno de programación específico**, y es a la vez **suficientemente estructurado** para que su implementación se pueda hacer directamente a partir de él.

- Es independiente del lenguaje de programación.

- La definición de datos se da por supuesta, principalmente para variables sencillas, pero si se emplea variable más complejas, por ejemplo pilas, colas, vectores, etc., se pueden definir en la cabecera del algoritmo.



```
Programa Encender lampara
si (lámpara enchufada) entonces
    si (lampara encendida) entonces
        problema solucionado
    si no
        si (foco quemado) entonces
            reemplazar foco
        si no
            comprar nueva lámpara
        fin si
    fin si
si no
    enchufar lámpara
        si (lampara encendida) entonces
            problema solucionado
        si no
            si (foco quemado) entonces
                reemplazar foco
            si no
                comprar nueva lámpara
            fin si
        fin si
fin si
```

Convenciones comunes de un pseudocódigo

- **Asignación**

$x \leftarrow y$

$y \rightarrow x$

- **Variables declaradas por el desarrollador**

$volumen \leftarrow \prod r^2 h$

$resultado \leftarrow sin(a)$

- **Estructuras de control**

Secuencia de instrucciones

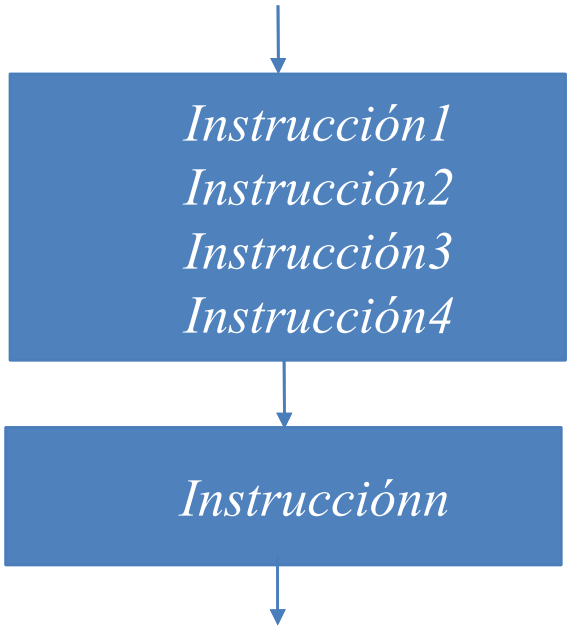


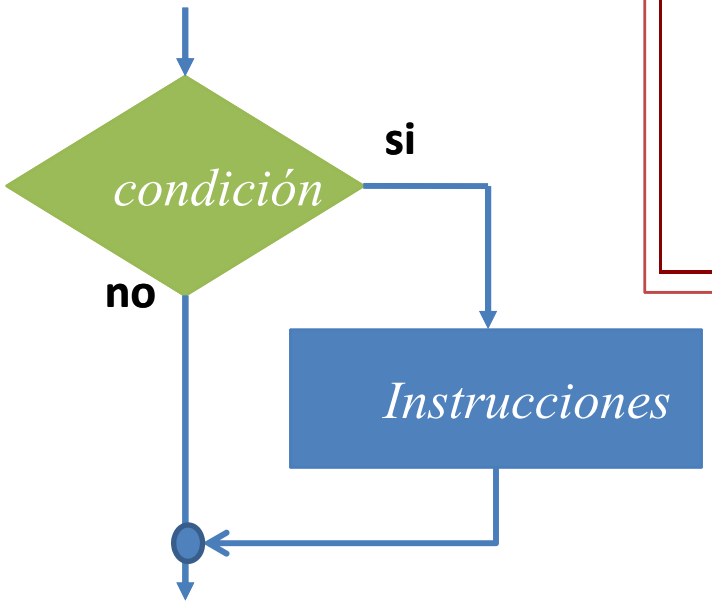
Diagrama de flujo

$$instrucción_1$$
$$instrucción_2$$
$$instrucción_3$$
$$\ldots$$
$$instrucción_n$$

Pseudocódigo

- **Estructuras de control**

Condicionales

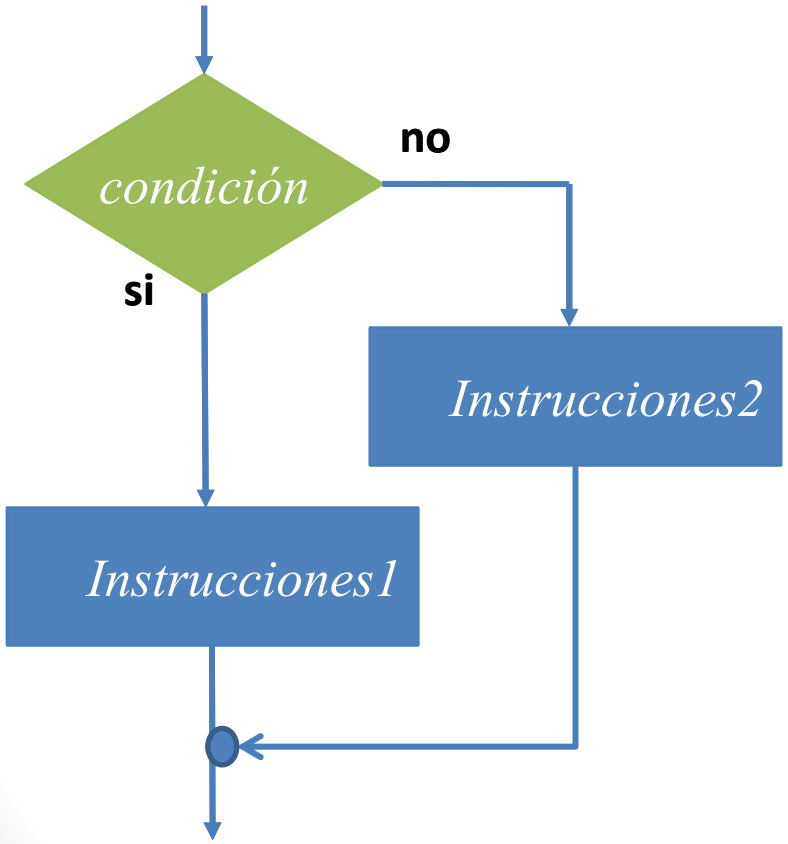



```
si condición entonces
    instrucciones
fin si
```

Pseudocódigo

- **Estructuras de control**

Condicional doble

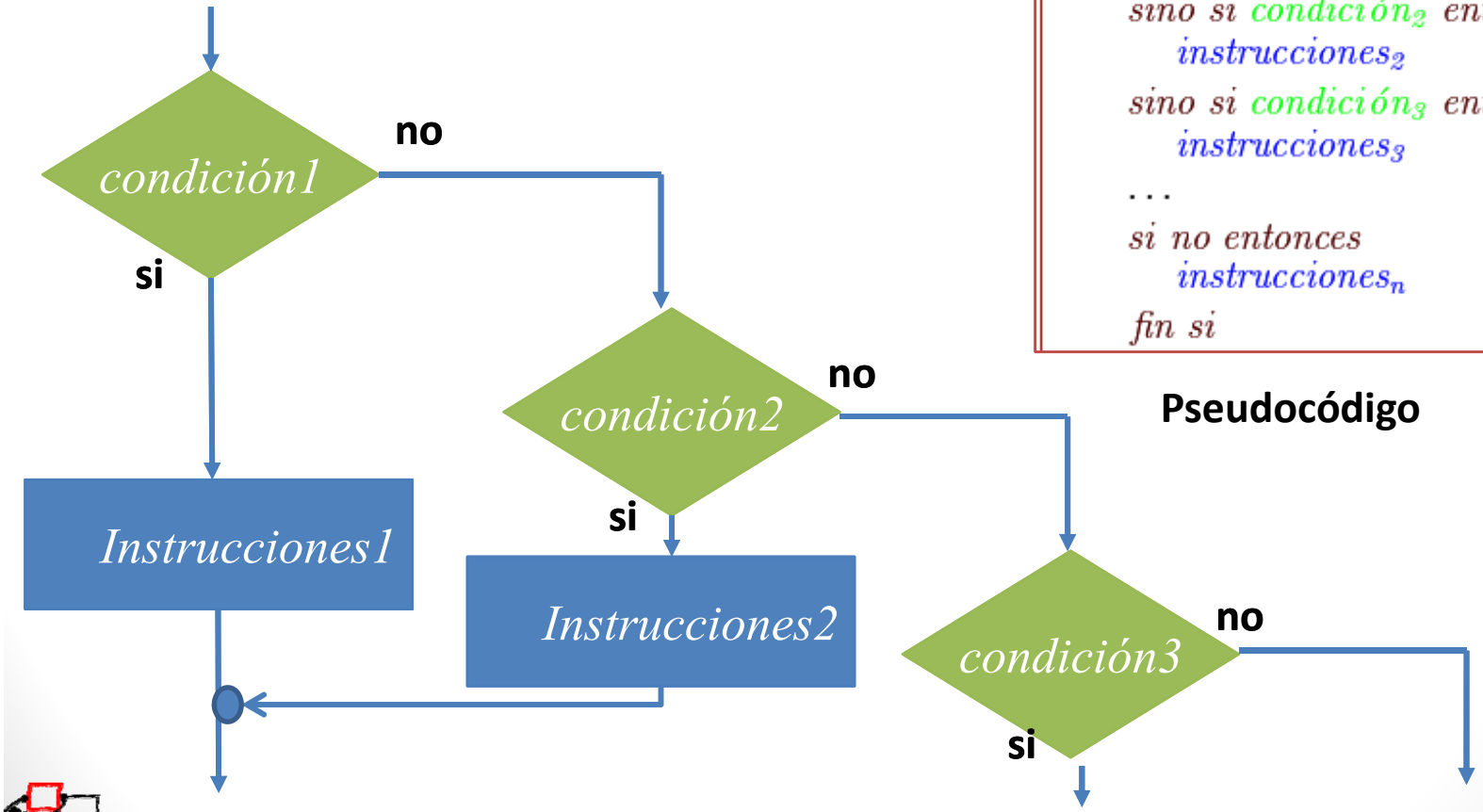



```
si condición entonces
    instrucciones1
si no entonces
    instrucciones2
fin si
```

Pseudocódigo

- **Estructuras de control**

Condicional múltiple

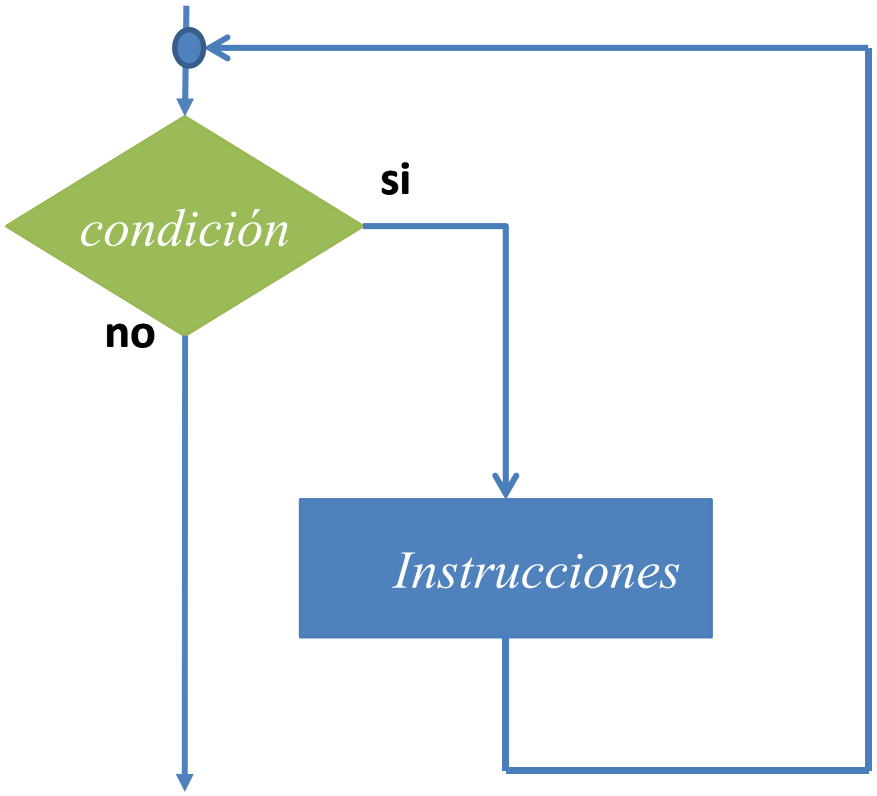


si $condición_1$ *entonces*
  $instrucciones_1$
sino si $condición_2$ *entonces*
  $instrucciones_2$
sino si $condición_3$ *entonces*
  $instrucciones_3$
...
si no entonces
  $instrucciones_n$
fin si

Pseudocódigo

- **Estructuras de control**

Iterativa

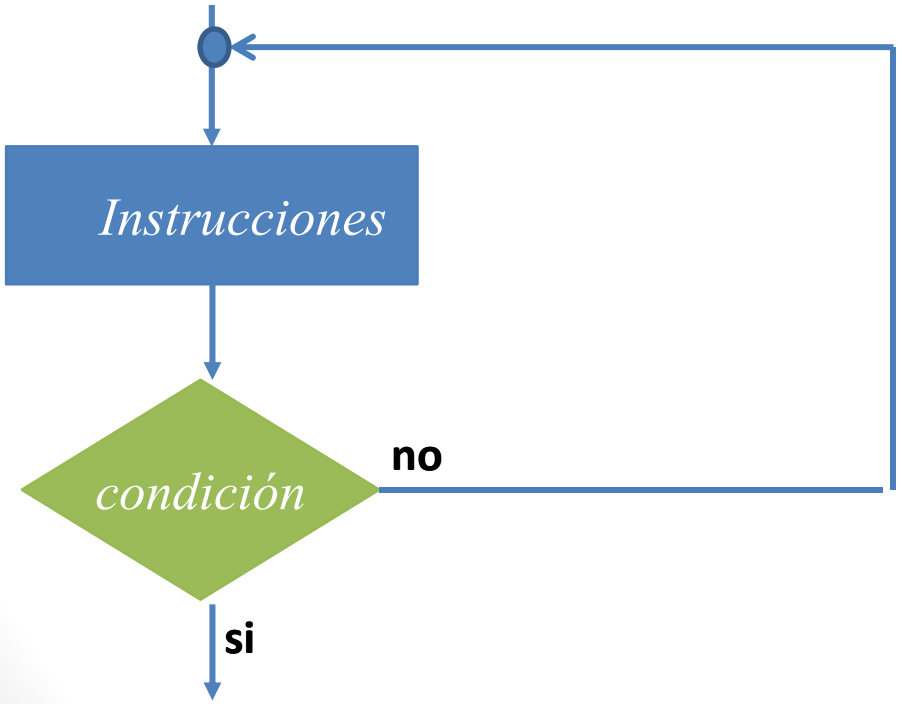



```
mientras condición hacer
    instrucciones
fin mientras
```

Pseudocódigo

Estructuras de control

Iterativa



Pseudocódigo

```
repetir
    instrucciones
hasta que condición
```

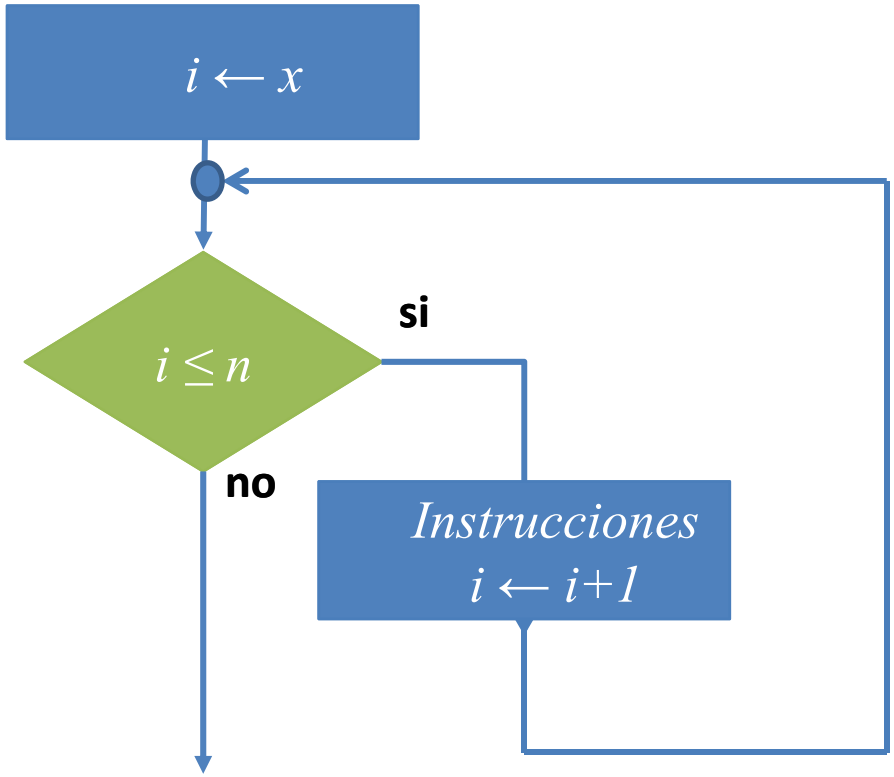
Repetir

```
instrucciones
mientras ¬(condición) hacer
    instrucciones
fin mientras
```

Mientras ≈ Repetir

Estructuras de control

Iterativa



Pseudocódigo

```
para i ← x hasta n hacer
    instrucciones
fin para
```

Para

```
i ← x
mientras i ≤ n hacer
    instrucciones
    i ← i + 1
fin mientras
```

Para → mientras

Anidamiento

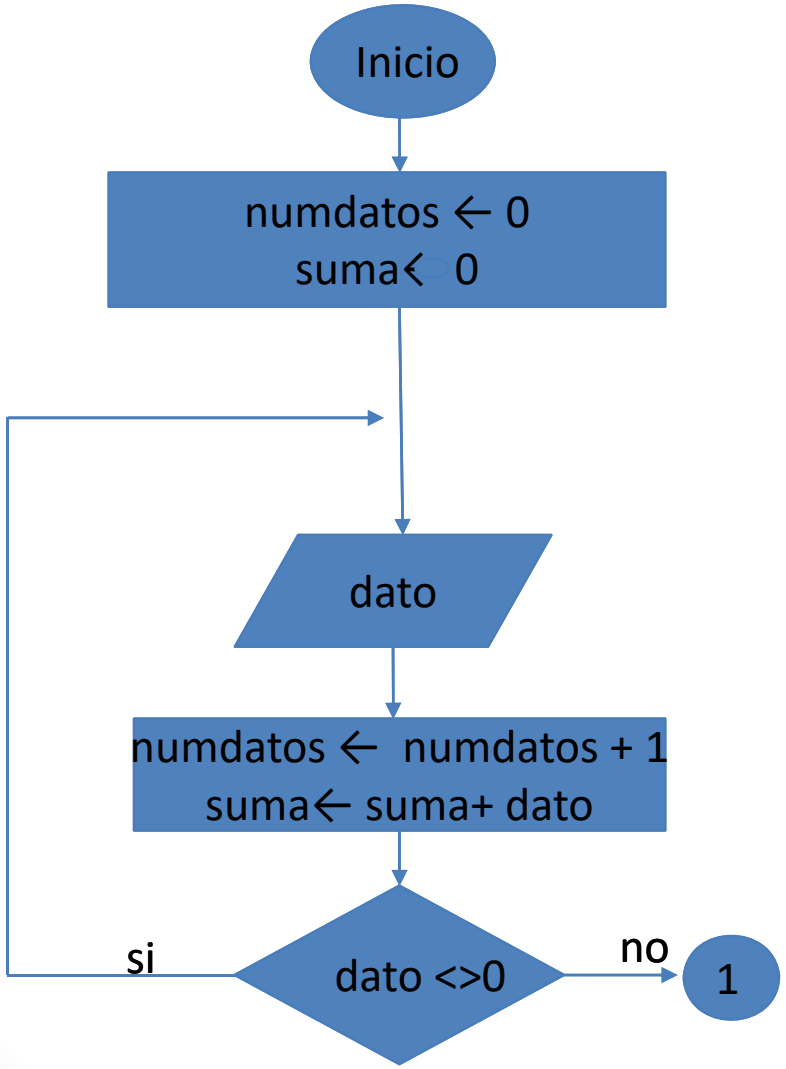
```
procedimiento Ordenar (L)
    //Comentario: L = (L_1, L_2, ..., L_n) es una lista con n elementos//
    k ← 0
    repetir
        intercambio ← falso
        k ← k + 1
        para i ← 1 hasta n − k hacer
            si L_i > L_{i+1} entonces
                intercambiar (L_i, L_{i+1})
                intercambio ← verdadero
            fin si
        fin para
    hasta que intercambio = falso
fin procedimiento
```

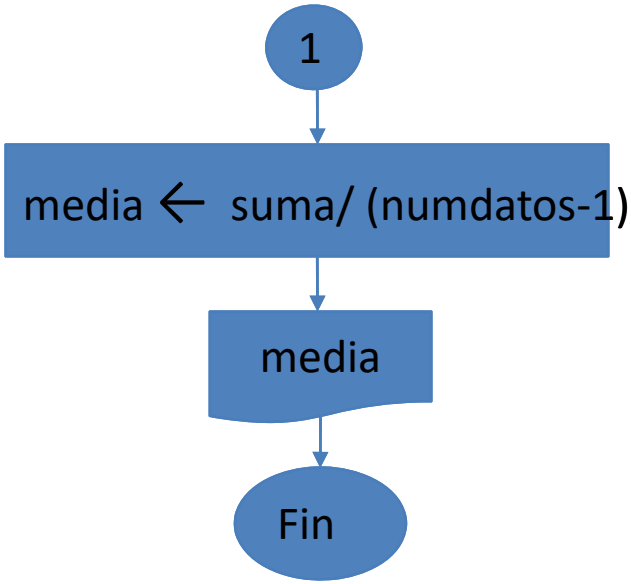
Ejemplo 04

- Calcular la media de una serie de números positivos, suponiendo que los datos se leen uno a uno.

- Un valor de cero como entrada indicará que se ha alcanzado el final de la serie de números positivos.

Ejemplo 04 (Algoritmo en diagrama de flujo)





Ejemplo 04 (Algoritmo en pseudocódigo)

```
Procedimiento Media()
    numdatos<-0
    suma<-0
    repetir
        dato<-Entrada()
        numdatos<-numdatos+1
        suma->suma+dato
    hasta que ¬(dato<>0)
    media<-suma/(numdatos-1)
    media->Salida()
fin procedimiento
```

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Academia de Ciencias de la Computación

Autor: M. en C. Edgardo Adrián Franco Martínez

Algoritmia y programación estructurada

Unidad I "Conceptos básicos y herramientas de programación"

1.2 La arquitectura de Von Neumann

Contenido

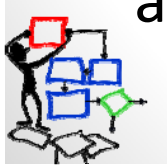
- Arquitectura de una computadora
  - Elementos básicos de una arquitectura
  - Aspectos que definen y distinguen una arquitectura
- Arquitectura de una computadora según la organización de los elementos
  - Arquitectura Harvard
  - Arquitectura Von Neumann
    - Funcionamiento general de este tipo de arquitectura

Arquitectura de una computadora

- **Computadora:** *"Máquina capaz de procesar información a muy alta velocidad".*

- Podemos determinar con esta definición que esta tiene una arquitectura establecida y un modo de funcionamiento, debido al hecho de ser una maquina.

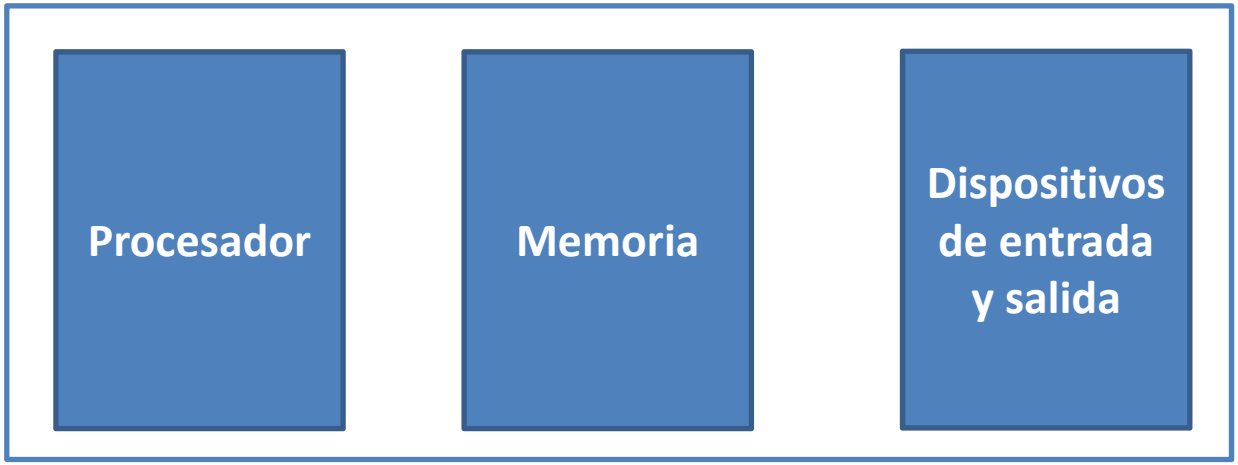
- La arquitectura de una computadora es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso CPU trabaja internamente y accede a las direcciones de memoria.



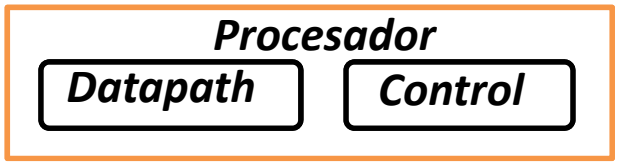
Elementos básicos de una arquitectura

- Todas las computadoras constan principalmente de tres partes, la **CPU** que procesa los datos, la **memoria** que guarda los datos y los **dispositivos de entrada y salida** que permiten la comunicación con el exterior.

Computadora digital



Procesador



- Desde el punto de vista funcional, un microprocesador es un circuito integrado que incorpora en su interior una unidad central de proceso (CPU) y todo un conjunto de elementos lógicos que permiten enlazar otros dispositivos como memorias y puertos de entrada y salida (I/O), formando un sistema completo para cumplir con una aplicación específica dentro del mundo real. Para que el sistema pueda realizar su labor debe ejecutar paso a paso un programa que consiste en una secuencia de números binarios o instrucciones, almacenándolas en uno o más elementos de memoria, generalmente externos al mismo

Memoria

- Se refiere a los componentes de una computadora, dispositivo y medios de almacenamiento que retienen datos informáticos durante algún intervalo de tiempo. Las memorias de computadora proporcionan unas de las principales funciones de la computación moderna, la retención o almacenamiento de información. Es uno de los componentes fundamentales de todas las computadoras modernas.

- La **memoria primaria** está directamente conectada a la CPU de la computadora. Debe estar presente para que la CPU funcione correctamente. *(Registros del procesador, Memoria cache y memoria principal de acceso aleatorio RAM).*

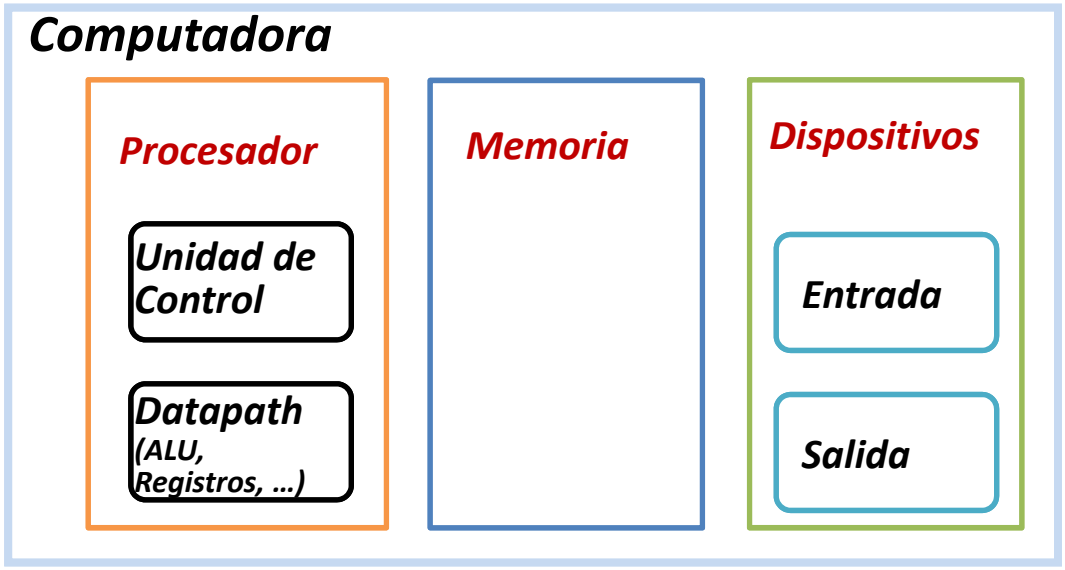
- La **memoria secundaria** requiere que la computadora use sus canales de entrada/salida para acceder a la información y se utiliza para almacenamiento a largo plazo de información persistente. *(Discos Duros, Memorias Flash, etc.)*

Memoria

Dispositivos de entrada-salida

- E/S o I/O (input/output), es la colección de interfaces que usan las distintas unidades funcionales (subsistemas) de un sistema de procesamiento de información para comunicarse unas con otras.

- De acuerdo a lo anterior, una arquitectura de computadora será formada por los siguientes elementos básicos.



*Un **datapath** es una colección de unidades funcionales, por ejemplo ALUs o multiplicadores, o unidades que realizan un proceso u operaciones con los datos. La mayoría de los procesadores consisten en un datapath y una a unidad de control, la unidad de control se dedica a regular la interacción entre el datapath y la memoria.*

Aspectos que definen y distinguen una arquitectura

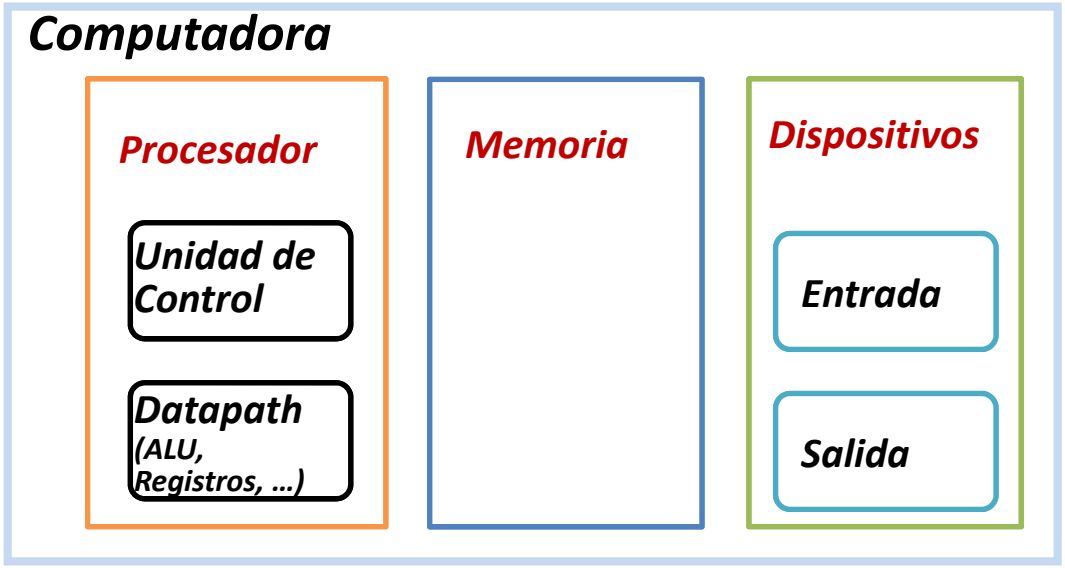
- Toda arquitectura computacional incluye tres **aspectos que la definen y distinguen**.

1. **Conjunto de operaciones**
2. **Organización de la computadora**
3. **Hardware de la computadora**

1. **El conjunto de operaciones:** es la interfaz visible entre el hardware y la programación.
   - Las dos principales aproximaciones al conjunto de instrucciones son:
     - ***CISC (Complex Instruction Set Computer)***
     - ***RISC (Reduced Instruction Set Computer)***

2. **La organización de la computadora:** es la lógica de funcionamiento de la arquitectura, pueden distinguirse dos arquitecturas teóricas básicas.

1. ***Arquitectura Von Neumann***
2. ***Arquitectura Harvard***



3. **El Hardware de la computadora:** es lo que físicamente lleva a cabo el trabajo de procesamiento. De acuerdo a las **capacidades y tipos** se organizan de acuerdo a una arquitectura estándar para la construcción de una computadora.

   - *i.e.* este aspecto se refiere a las características del hardware *(Velocidad, capacidad, …)*.

Arquitectura de una computadora según la organización de los elementos

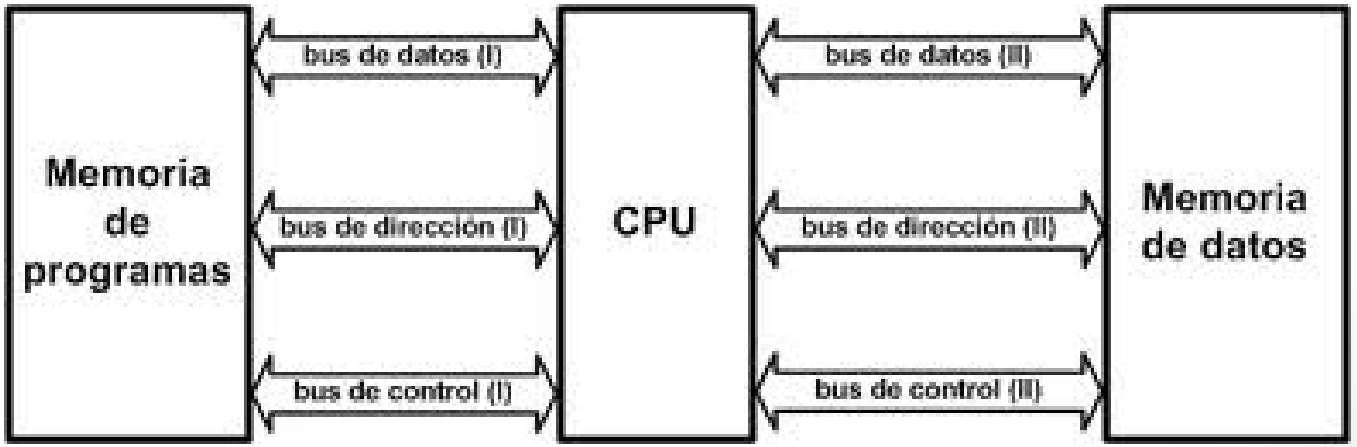
- **La organización de la computadora:** es la lógica de funcionamiento de la arquitectura, pueden distinguirse dos arquitecturas teóricas básicas.

1. ***Arquitectura Von Neumann***
2. ***Arquitectura Harvard***

- Ambos modelos contemplan la existencia de un modulo de procesamiento, una serie de dispositivos de entrada/salida y memoria.

Arquitectura Harvard

- Arquitectura Harvard hace referencia una organización de la computadora que utiliza dispositivos **memorias físicamente separadas para las instrucciones y para los datos.**
  - El término proviene de la computadora Harvard Mark I, que almacenaba las instrucciones en cintas perforadas y los datos en interruptores.



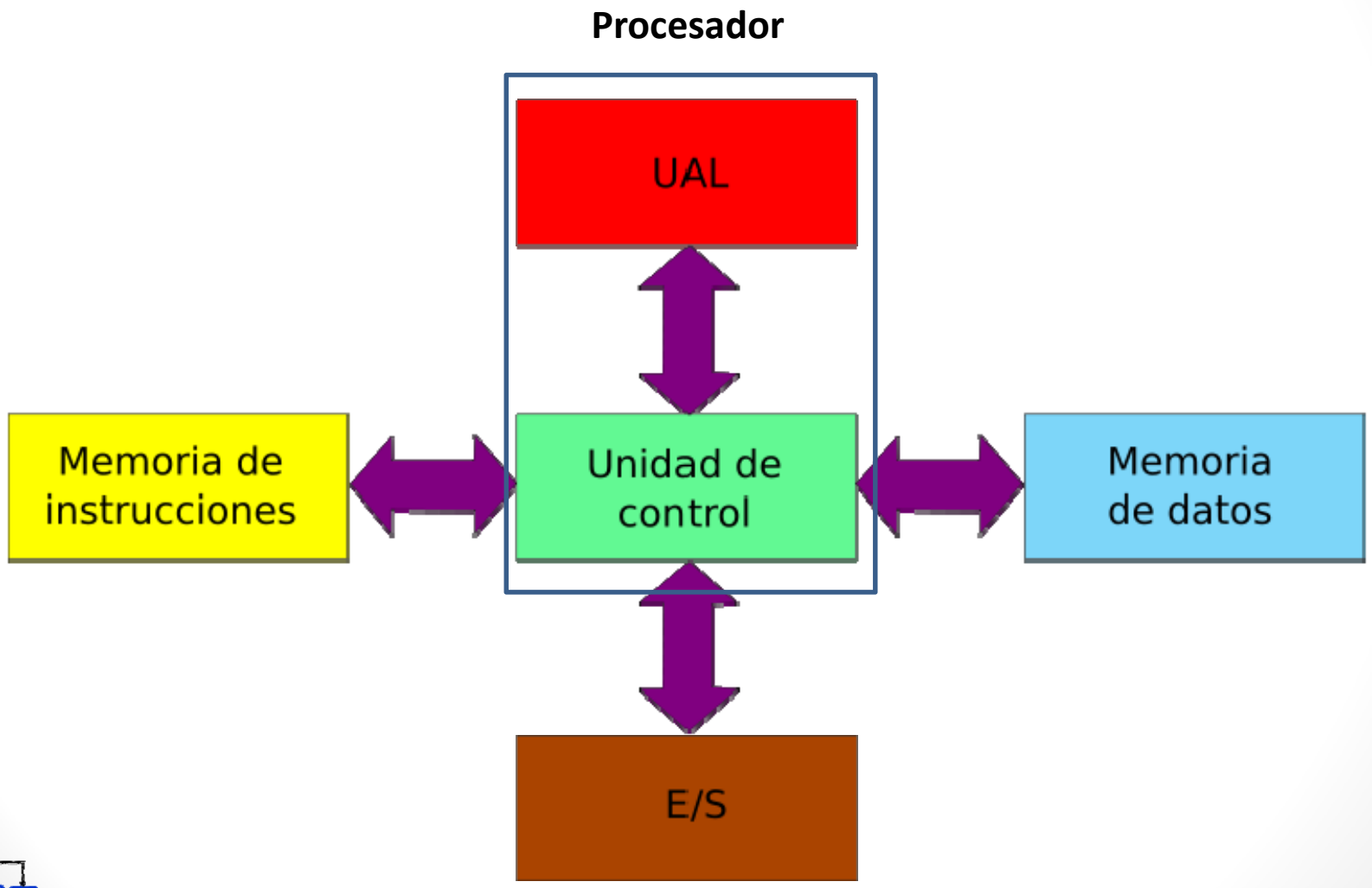
Memoria

- Cada memoria dispone de su respectivo bus, lo que permite, que la CPU pueda acceder de forma independiente y simultánea a la memoria de datos y a la de instrucciones.
- Como los buses son independientes éstos pueden tener distintos contenidos en la misma dirección .
- Además de que el ancho de palabra del bus de datos de cada memoria puede ser distinto.

Usos de esta arquitectura

- Esta arquitectura suele utilizarse en Microcontroladores y DSPs (procesadores digitales de señales), usados habitualmente en productos para procesamiento de audio y video así como sistemas electrónicos con cómputo embebido.

Organización de la computadora digital según la arquitectura Harvard



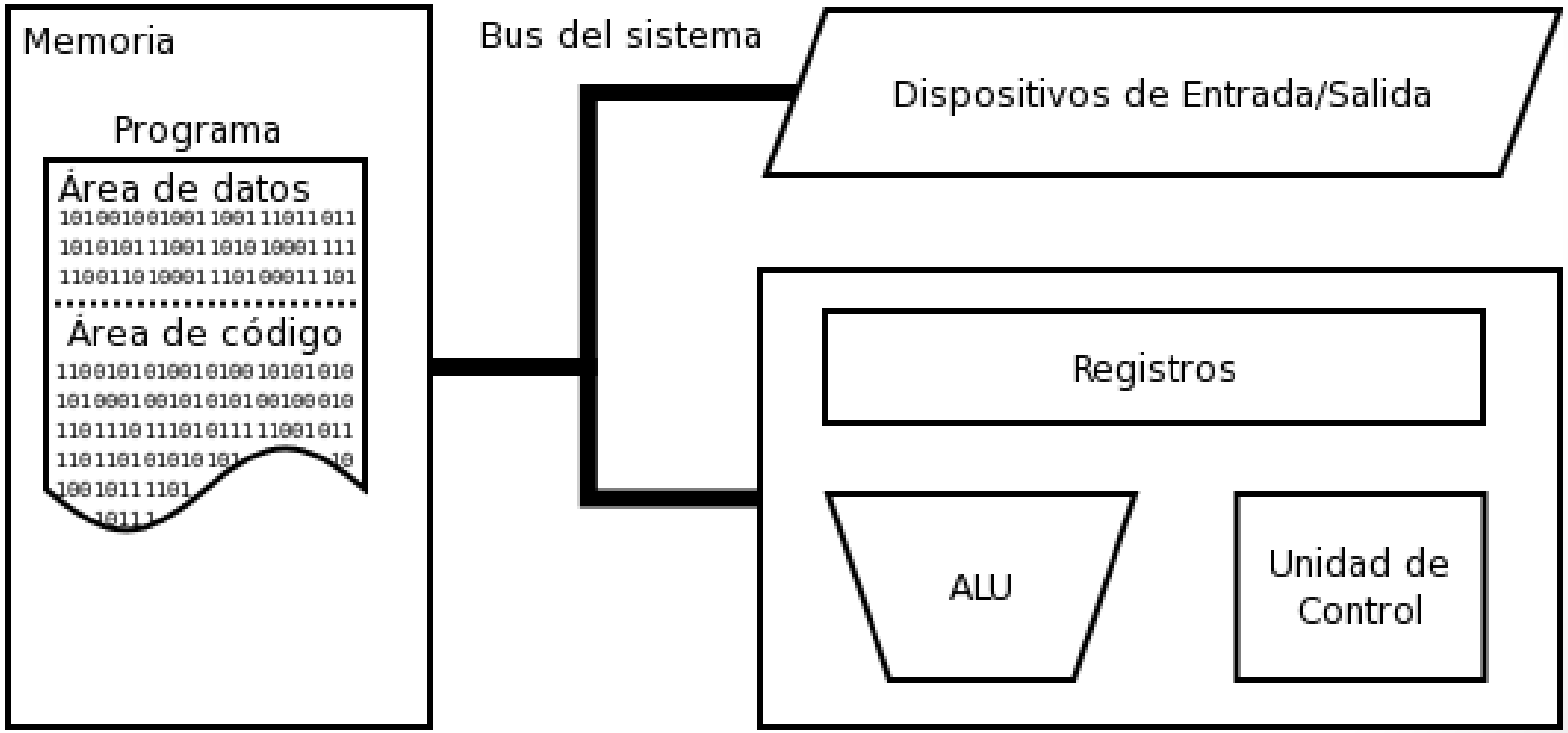
Arquitectura Von Neumann

- El nacimiento u origen de la arquitectura Von Neumann surge a raíz de una colaboración en el proyecto ENIAC del matemático de origen húngaro, John Von Neumann.

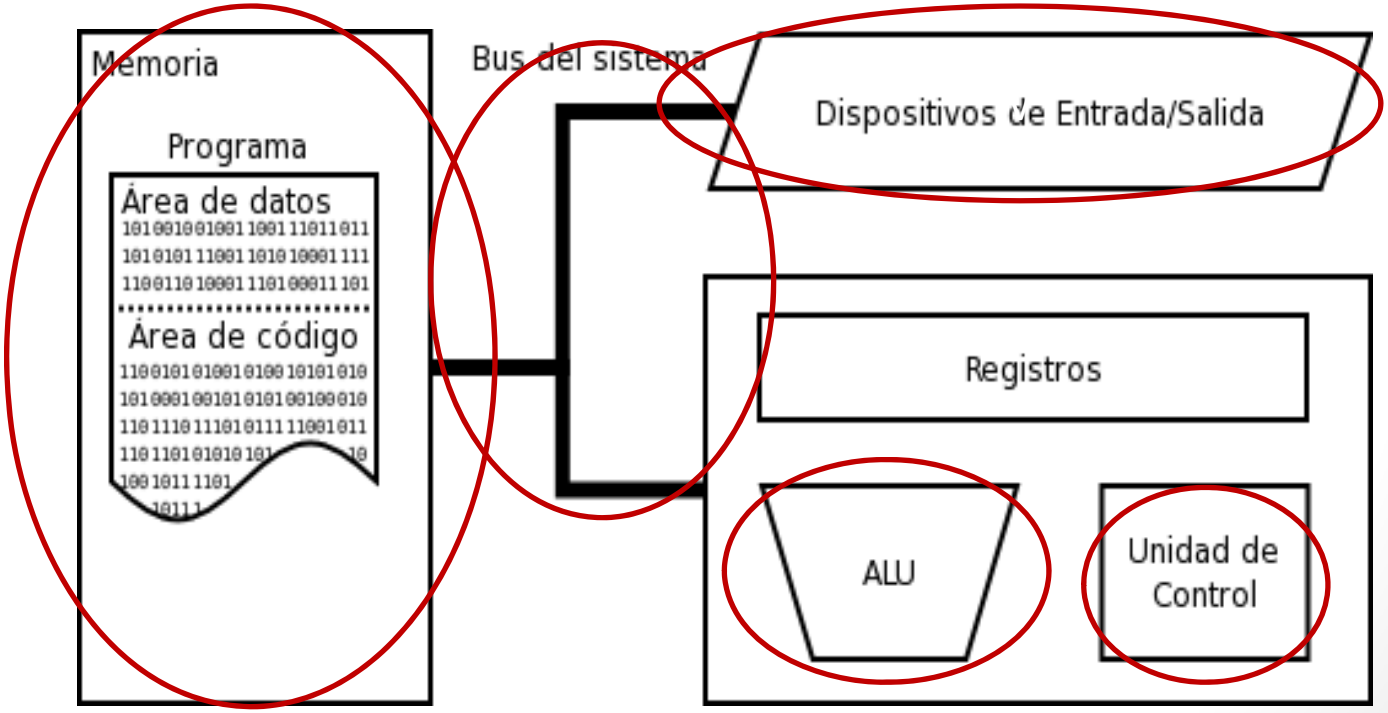
- Este trabajaba en 1945 en el laboratorio atómico de Los Alamos cuando se encontró con uno de los constructores de la ENIAC. Compañero de Albert Einstein, Goedel y Turing en Princeton, Von Neumann se interesó por el problema de la necesidad de "recablear" la máquina para cada nueva tarea.

- En 1949 había encontrado y desarrollado la solución a este problema, consistente en poner la información sobre las operaciones a realizar en la misma memoria utilizada para los datos, escribiéndola de la misma forma, es decir en código binario (Computadora EDVAC).

- La arquitectura Von Neumann es un modelo de organización en arquitecturas de computadoras que utilizan el **mismo dispositivo de almacenamiento** tanto **para las instrucciones como para los datos** *(a diferencia de la arquitectura Harvard)*.



- Los ordenadores con esta arquitectura constan de cinco partes: La ***unidad aritmético-lógica o ALU***, la ***unidad de control***, la ***memoria***, ***dispositivos de entrada/salida*** y el ***bus de datos*** que proporciona un medio de transporte de los datos entre las distintas partes.



Memoria

- Se compone de un **conjunto de celdas** del mismo tamaño (número de bits).
- Cada **celda** está **identificada** por un número binario único, denominado **dirección**.
- Una vez seleccionada una celda mediante su correspondiente dirección, se pueden hacer dos operaciones:
  - Lectura: Permite conocer el valor almacenado anteriormente.
  - Escritura: Almacena un nuevo valor.

Unidad Central de Proceso (CPU)

- Es el conjunto formado por la Unidad de Control, los registros y la Unidad Aritmética Lógica, es decir es el bloque encargado de ejecutar las instrucciones.
  - Con la aparición de los circuitos integrados, y en concreto a partir de los años 70, cuando la tecnología alcanzó el nivel de integración adecuado, se integró en una sola pastilla la CPU. A este circuito integrado se le denomina Microprocesador.

Una forma de determinar el rendimiento de un computador es por el número de instrucciones que ejecuta por segundo (MIPS).

Unidad Aritmético-Lógica (ALU)

- Realiza las operaciones elementales, tanto aritméticas como lógicas, que implementa el computador: suma, resta, AND, OR, NOT, etc.
- Los datos con los que opera se leen de la memoria, y pueden almacenarse temporalmente en los registros que contiene la CPU.

Unidad de Control

- Ejecuta las instrucciones máquina almacenadas en la memoria.
- Captura las instrucciones y las decodifica.
- Según el tipo de instrucción, genera las señales de control a todas las unidades internas de la CPU para poder realizar su ejecución.

Unidad de Entrada/Salida

- Realiza la transferencia de información con las unidades externas, denominadas periféricos: unidades de almacenamiento secundario (disco duro, disquete, cinta, etc.), impresoras, terminales, monitores, etc.

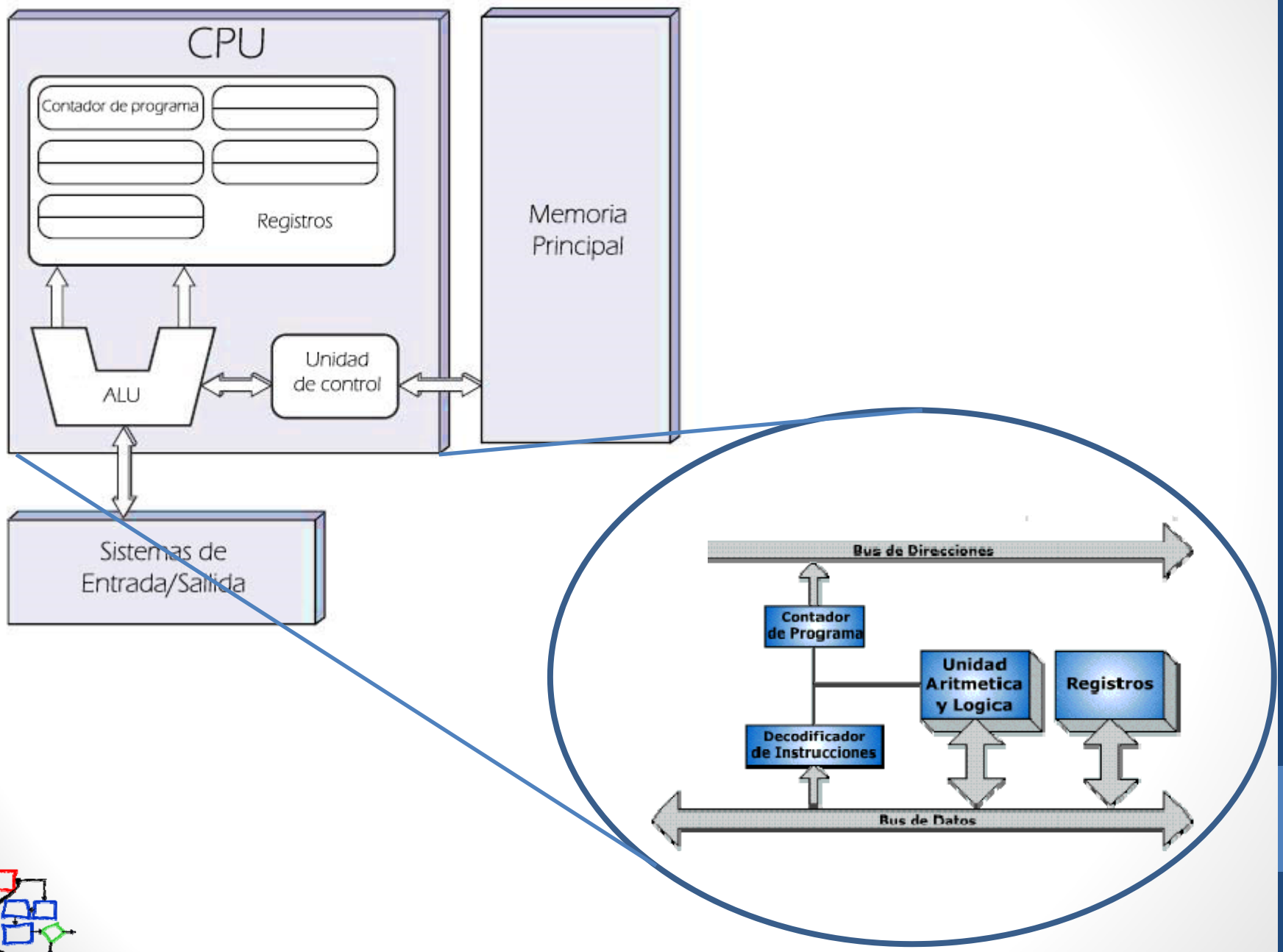
- La memoria secundaria (MS), se considera como un periférico. La MS es más lenta que la principal, pero tiene una mayor capacidad de almacenamiento.

Buses

- Además de las 4 unidades básicas, en un computador existen conjuntos de señales, que se denominan buses, y cuya función es transferir las instrucciones y los datos entre las distintas unidades.

- Estos buses se representan en la figura mediante flechas de trazo continuo. Se suelen distinguir tres tipos de buses:

  - *Bus de direcciones*
  - *Bus de datos*
  - *Bus de control*


CPU
Contador de programa
Registros
ALU
Unidad de control
Memoria Principal
Sistemas de Entrada/Salida
Bus de Direcciones
Contador de Programa
Unidad Aritmetica y Logica
Registros
Decodificador de Instrucciones
Bus de Datos

Funcionamiento general de este tipo de arquitectura

- Un ordenador con esta arquitectura realiza o emula los siguientes pasos secuencialmente:

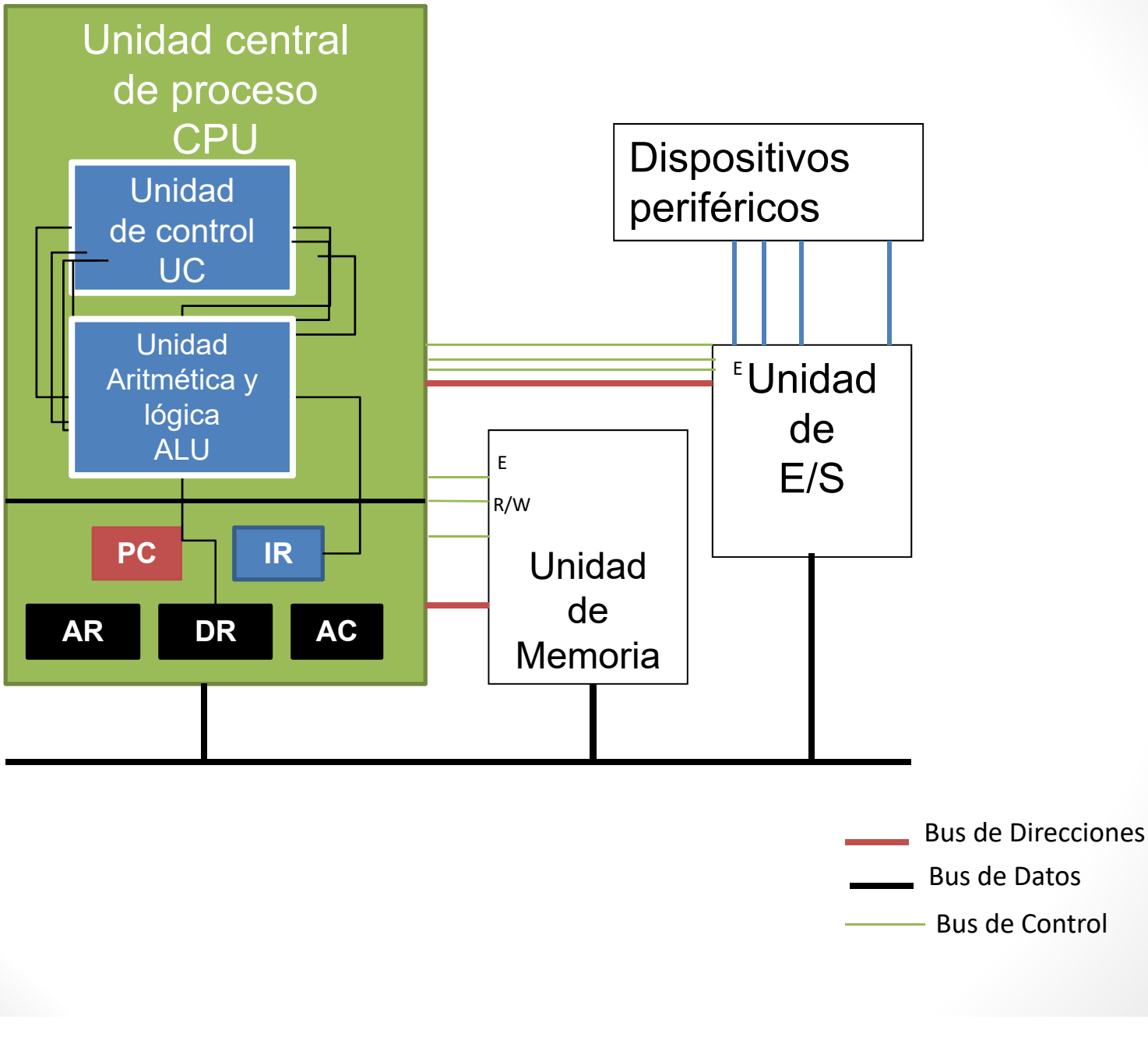
1. Obtiene la siguiente instrucción desde la memoria en la dirección indicada por el **contador de programa (PC)** y la guarda en el **registro de instrucción (IR)**.

2. Aumenta el contador de programa en la longitud de la instrucción para apuntar a la siguiente instrución.

3. **Decodifica** la instrucción mediante la **unidad de control**. Ésta se encarga de coordinar el resto de componentes del ordenador para realizar una función determinada.

4. **Se ejecuta la instrucción** en este paso puede cambiar el valor del contador del programa, permitiendo así operaciones repetitivas. El contador puede cambiar también cuando se cumpla una cierta condición aritmética, haciendo que el ordenador pueda 'tomar decisiones', que pueden alcanzar cualquier grado de complejidad, mediante la aritmética y lógica anteriores.

5. **Vuelve al paso 1**


Unidad central de proceso CPU
Unidad de control UC
Unidad Aritmética y lógica ALU
PC
IR
AR
DR
AC
Dispositivos periféricos
E
Unidad de E/S
E
R/W
Unidad de Memoria
Bus de Direcciones
Bus de Datos
Bus de Control

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Academia de Ciencias de la Computación

Autor: M. en C. Edgardo Adrián Franco Martínez

Algoritmia y programación estructurada

Unidad I "Conceptos básicos y herramientas de programación"

1.3 Herramientas de programación

Contenido

- Lenguaje de programación
- Programa computacional
- Clasificaciones de los lenguajes de programación
  - Clasificación según su nivel de abstracción
  - Clasificación según su modo de ejecución final
  - Clasificación según su paradigma de programación
- Lenguaje C
  - Historia del lenguaje C
  - Ventajas y desventajas del lenguaje C
  - El estándar ANSI C

Lenguaje de programación

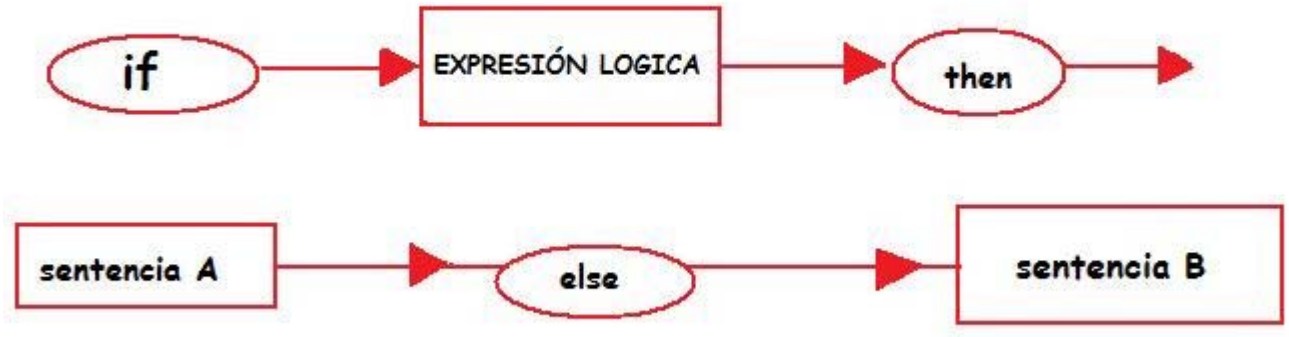
- Un lenguaje de programación es un **idioma artificial diseñado para expresar computaciones** que pueden ser llevadas a cabo por máquinas como las computadoras.

- Pueden usarse para **crear programas que controlen el comportamiento físico y lógico de una máquina**, esto **permite expresar algoritmos** con precisión e interacción humano-maquina.

- **Está formado de un conjunto de *símbolos* y *reglas sintácticas* y *semánticas*** que definen su estructura y el significado de sus elementos y expresiones.

  - ***Sintáctica*** *(reglas que gobiernan la combinatoria de los símbolos y la formación de unidades superiores a estos)*

  - ***Semántica*** *(aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal)*



Programa computacional

- Un programa computacional es un **conjunto de instrucciones** que **una vez ejecutadas realizarán una o varias tareas en una computadora**.

- La razón de ser de un programa computacional es permitir resolver problemas con apoyo de equipos computacionales por lo que al crearlos es necesario **abstraer** los componentes de este y modelarlos en un ambiente computacional.

Abstracción:, acto mental en el que conceptualmente se aísla un objeto o una propiedad de un objeto.

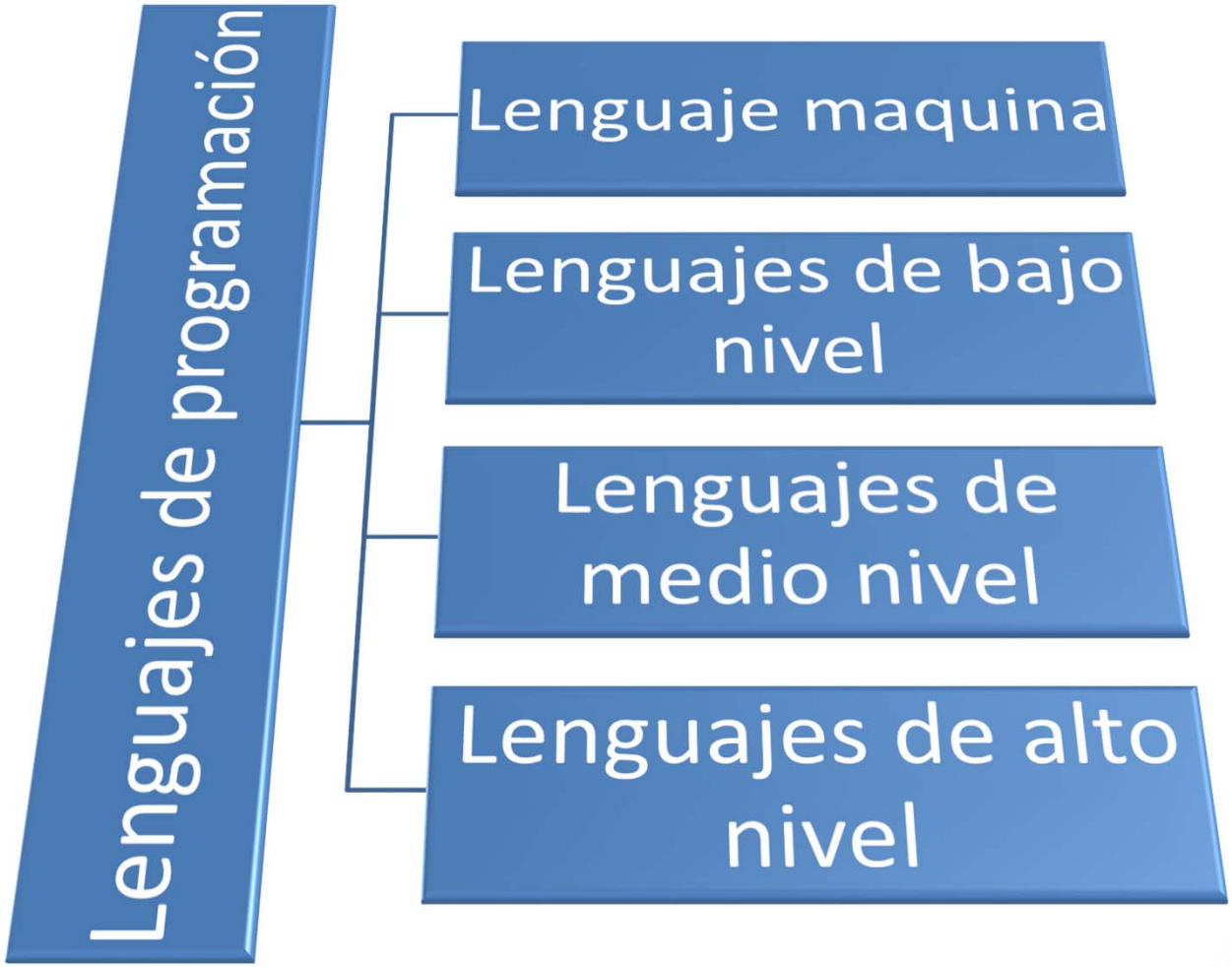
Clasificación de los lenguaje de programación

- En la actualidad hay muchos tipos de lenguajes de programación, cada uno de ellos con sus propias reglas, terminología, sintaxis y manera de crear un programa computacional.

- La clasificación de los lenguajes de programación puede realizarse desde tres aspectos básicos.

  - Según su nivel de abstracción
  - Según su modo de ejecución final
  - Según su paradigma de programación

Clasificación de los lenguaje de programación según su nivel de abstracción



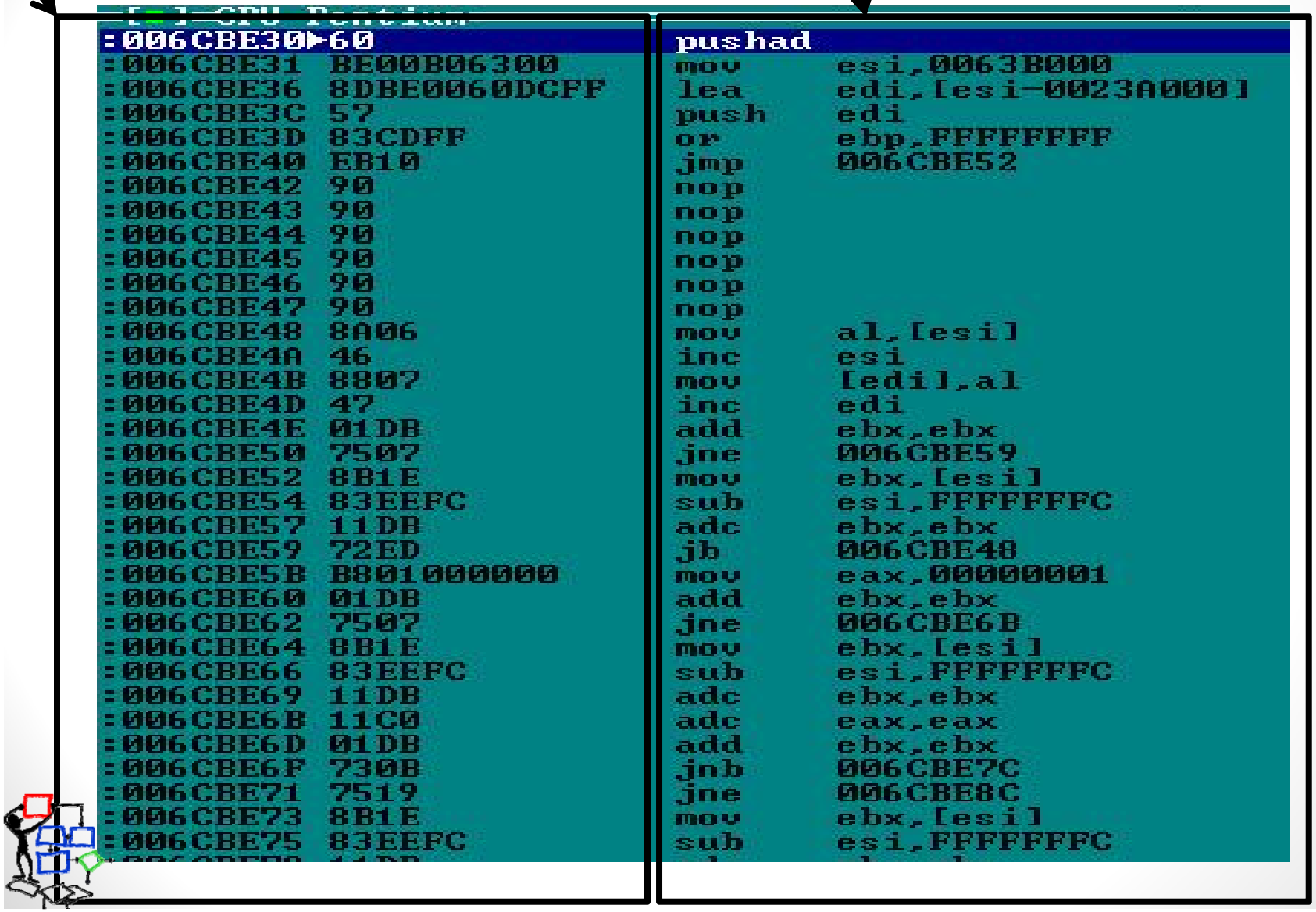
- **Lenguaje Máquina:** es el lenguaje de programación que entiende directamente la computadora o máquina. Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1.

- **Lenguajes de programación de bajo nivel:** Son mucho mas fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la máquina o computadora al igual que el lenguaje máquina.

  - El lenguaje ensamblador fue el primer lenguaje de programación que trato de sustituir el lenguaje máquina por otro mucho más parecido al de los seres humanos.

Lenguaje de bajo nivel
(Instrucciones en código maquina)

Lenguaje maquina (Instrucciones en binario)



- **Lenguaje de medio nivel:** Lenguaje de medio nivel es un lenguaje de programación que se encuentran entre los lenguajes de alto nivel y los lenguajes de bajo nivel. **Suelen ser clasificados muchas veces de alto nivel,** pero permiten ciertos manejos de bajo nivel.

- Son precisos para ciertas aplicaciones como la creación de sistemas operativos, ya que permiten un manejo abstracto (independiente de la máquina, a diferencia del ensamblador), pero sin perder mucho del poder y eficiencia que tienen los lenguajes de bajo nivel.

ejemplo C: Hola Mundo!

```
#include <stdio.h>

int main()
{
  printf("Hola Mundo!\n");
  return 0;
}
```

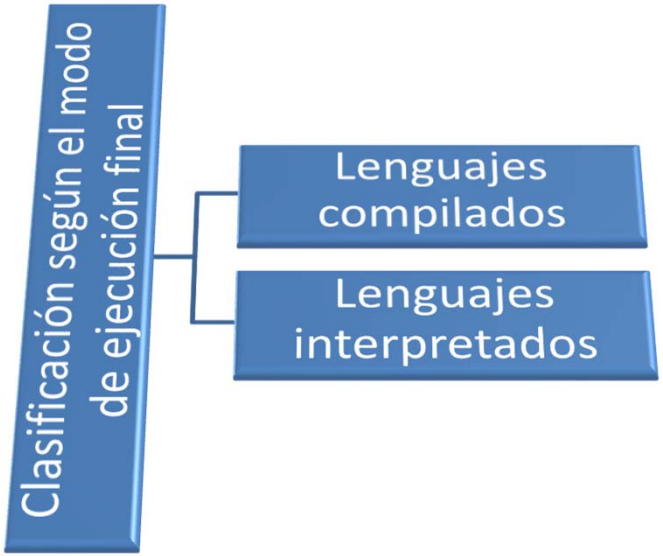
```
int num_a = 34;
int num_b = 14;
int result;

void main() {
  result = num_a * num_b;
}
```

```
; ADDRESS

$0000   MOVLW     128
GOTO _m $000A
$005D
$005D   $000B   BTFSS
MOVLW        CL $001F
$005E   $000C   GOTO $+10
BCF  ST      BT $0020
$005F   $000D   MOVF STACK
BCF  ST      GC $0021
$0060   $000E   ADDWF
MOVWF        CC $0022
$0061   $000F   MOVF STACK
MOVLW        CC $0023
$0062   $0010   BTFSC
MOVWF        IN $0024
$0063   $0011   INCFSZ
MOVLW        BT $0025
$0064   $0012   ADDWF
MOVWF        IN $0026
$0065   $0013   BTFSC
MOVLW        IN $0027
$0066   $0014   INCF STACK
MOVWF        BT $0028
$0067   $0015   BCF  STATU
RETURN       GC $0029
$0004   $0016   BTFSS
$0004        CC $002A
BCF  ST $0017   GOTO $+7
$0005        CC $002B
BCF  ST $0018   MOVF STACK
$0006        IN $002C
        $0019   ADDWF
             BT $002D
                BTFSC
                $002E

                $001E     $1C03
                STATUS, C
                $0033     $0CF9
                RRF  STACK_9, F
                $0034     $0CF8
                RRF  STACK_8, F
                $0035     $1C03
                BTFSS     STATUS, C
                $0036     $281C
                GOTO $-26
                $0037     $1C7D
                BTFSS     STACK_13, 0
                $0038     $2844
                GOTO $+12
                $0039     $09FB
                COMF STACK_11, F
                $003A     $09FA
                COMF STACK_10, F
                $003B     $09F9
                COMF STACK_9, F
                $003C     $09F8
                COMF STACK_8, F
                $003D     $0AF8
                INCF STACK_8, F
                $003E     $1903
                BTFSC     STATUS, Z
                $003F     $0AF9
                INCF STACK_9, F
                $0040     $1903
                BTFSC     STATUS, Z
                $0041     $0AFA
                INCF STACK_10, F
                $0042     $1903
                BTFSC     STATUS, Z
                $0043     $0AFB
```

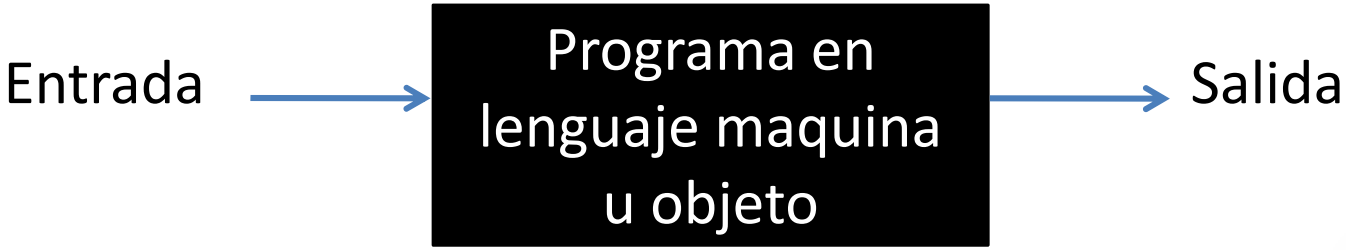
- **Lenguajes de programación de alto nivel:** Este tipo de lenguajes de programación son independientes de la máquina, lo podemos usar en cualquier computador con muy pocas modificaciones o sin ellas, son muy similares al lenguaje humano, pero precisan de un programa interprete o compilador que traduzca este lenguaje de programación de alto nivel a lenguaje de máquina que la computadora pueda entender.

Clasificación de los lenguaje de programación según su modo de ejecución final

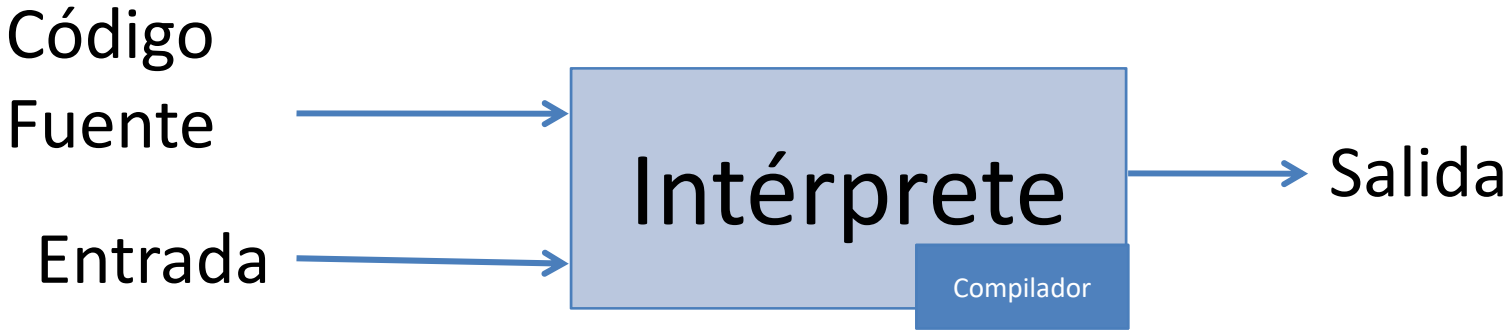
- El **modo de ejecución final** de un lenguaje de programación, se refiere al **proceso necesario** para **poner en ejecución las instrucciones** de dicho lenguaje en un equipo de cómputo. Para finalmente proporcionar las entradas que serán tomadas para obtener una salida de todo el conjunto de instrucciones (programa computacional).



- **Lenguaje compilado**
  - Lenguaje de programación que requiere de un proceso de compilación antes de poder ser ejecutado.



- **Lenguaje interpretado**
  - Es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete.



Clasificación de los lenguaje de programación según su paradigma de programación

- El **paradigma de programación** es un modelo que rige como construir un programa de computación bajo un lenguaje de programación, por lo que algunos lenguajes han surgido orientándose a ellos. También existen lenguajes de programación capaces de soportar más de un paradigma de programación

Un paradigma de programación representa un enfoque particular o **filosofía para la construcción del software**.

- Un paradigma de programación no es mejor uno que otro sino que **cada uno tiene ventajas y desventajas**.

- También hay situaciones donde un paradigma resulta más apropiado que otro.

- En la actualidad el paradigma **orientado a objetos es el más utilizado** debido a la facilidad para abstraer a su filosofía la mayoría de las soluciones a los problemas actuales, para su implementación y a reemplazado al paradigma de programación estructurada muy empleado en la década de los 80's y 90's.

Lenguaje C

- **C** es un **lenguaje de programación** creado en 1972 por **Dennis MacAlistair Ritchie** en los Laboratorios Bell como evolución del anterior lenguaje B, creado por Ken Thompson.

- Se trata de un **lenguaje fuertemente tipificado de medio nivel** pero con ***muchas características de bajo nivel***.

- Dispone de las **estructuras típicas** de los ***lenguajes de alto nivel*** pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel.

- Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

Ventajas del Lenguaje C

- **Programación Estructurada**
- Economía de expresiones
- Gran cantidad de operadores y tipos de datos
- Codificación en alto y bajo nivel simultáneamente
- Reemplaza ventajosamente la programación en ensamblador
- Utilización natural de las funciones primitivas del sistema operativo (Unix)
- De propósito general

Desventajas del Lenguaje C

- No posee de instrucciones de entrada y salida
- No posee de instrucciones de manejo de cadenas de caracteres
- La libertad en la escritura en los programas lleva a errores en la programación (semánticos) que, por ser correctos sintácticamente no se detectan a simple vista
- La precedencia de operadores convierten las expresiones en pequeños rompecabezas

Código fuente en C

- Un código fuente en C, es un conjunto de líneas que expresan computaciones bajo la sintaxis y semántica del lenguaje C. Un programa escrito en C tiene como características sobresalientes, el ser eficiente y veloz.

- Como el lenguaje C es compilado, se requiere del empleo de un compilador apropiado según la plataforma en la que se desea ejecutar el programa.

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Academia de Ciencias de la Computación

Autor: M. en C. Edgardo Adrián Franco Martínez

Algoritmia y programación estructurada

Unidad I "Conceptos básicos y herramientas de programación"

1.5 Representación de expresiones

Contenido

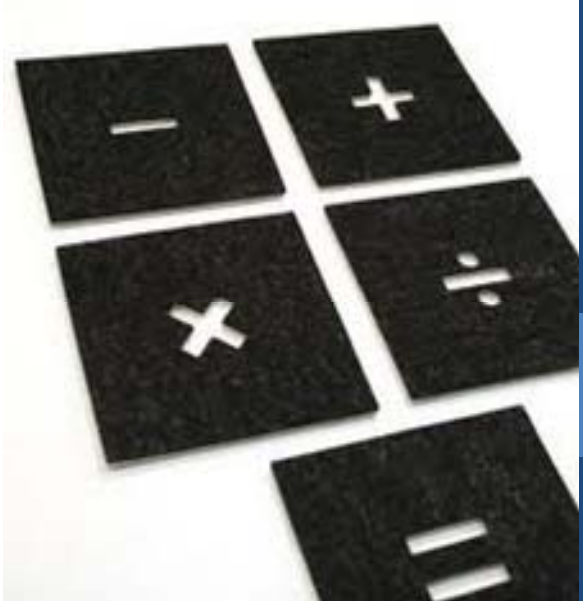
- Operadores
- Precedencia de Operadores
- Operadores y expresiones aritméticas
- Expresiones condicionales; operadores relacionales y lógicos
- Operaciones para el manejo de bits
- La librería estándar <math.h>
  - Funciones
  - Constantes
  - Ejemplo

Operadores

- Son **palabras o símbolos que implican una acción sobre ciertas variables**.

- Pueden operar de manera:
  - Unaria (con 1 variable)
  - Binaria (con 2 variables)
  - Ternaria (con 3 variables)

- Existen en C operadores:
  - Aritméticos
  - Relacionales
  - Lógicos
  - De Asignación
  - De Dirección
  - De Bits



Precedencia de Operadores

Operadores	Tipo	Asociatividad
( ) [ ] ->	Alta prioridad	Izquierda a derecha
! ~ ++ -- + - * & (tipo) sizeof	Unarios	Derecha a izquierda
* / % + -	Aritméticos	Izquierda a derecha
<< >>	Corrimiento de bits	Izquierda a derecha
< <= > >= == !=	Relacionales	Izquierda a derecha
& ^ |	Bits	Izquierda a derecha
&& ||	Lógicos	Izquierda a derecha
?:	Condicional	Derecha a izquierda
= += -= *= /= %= &= ^= |= <<= >>=	Asignación	Derecha a izquierda
,	Evaluación	Izquierda a derecha

Operadores y expresiones aritméticas

Operador	Nombre	Descripción
*	Multiplicación	5*2 → 10
/	División	5/2 → 2
%	Módulo	5%2 → 1
+	Suma	5+2 → 7
-	Resta	5-2 → 3
(tipo de dato)	“Cast” forzado	(double)5 → 5.0
sizeof(tipo de dato)	Tamaño de dato	sizeof(int) → 4

Operadores y expresiones de asignación

Operador	Abreviado	No Abreviado
=	a=2;	a=2;
++	n++;	n=n+1;
--	n--;	n=n-1;
+=	n+=2;	n=n+2;
-=	n-=2;	n=n-2;
*=	n*=2;	n=n*2;
/=	n/=2;	n=n/2;
%=	n%=2;	n=n%2;

Expresiones

- Son **expresiones validas** aquellas que utilicen de manera adecuada algún tipo de operador, y el resultado de su evaluación dependerá del orden de los operadores en la expresión, su prioridad y asociatividad.

Expresión en C	Significado
a=a+5*b;	a=a+(5*b)
x*=y+1;	x=x*(y+1)
x=++n;	x=n+1 y n=n+1
y=n--*5+n;	y=((n-1)*5)+n y n=n-1
z=n+1*45+10.4E-10*6	z=n+(1*45)+(10.4E-10*6)
z=(n+1)*(45+10.4E-10)*6	z=(n+1)*(45+10.4E-10)*6
x+=x*(10.5E11+x);	x=x+(x*10.5E11)+x
b=z=x=10.5;	b=10.5 y z=10.5 y x=10.5

- Son **expresiones aritméticas invalidas** las siguientes.

Expresión en C	Posible significado a dar
a=a+5*b=4;	a=a+(5*b) y b=4
x*=y+1++;	x=x*(y+2)
x=++n--;	x=n+1 y n=n-1
y=n--*5++n;	y=((n-1)*6)+n y n=n-1
z=n+1*45+10.4x10^-10*6	z=n+(1*45)+(10.4E-10*6)
z=(n+1)*(45+10.4E-10)*6=x	z=(n+1)*(45+10.4E-10)*6 y x=z
++x=x*(10.5E11+x++);	x=x+1+x+(x*10.5E11)+x;

Casting (Moldeado de variables)

- Debido a que el lenguaje C es un lenguaje fuertemente tipificado, es necesario estar consientes de las asignaciones que se realizan entre distintos tipos de datos.

- Aunque sintácticamente el lenguaje permite realizar asignaciones entre variables de distintos tipos de datos es necesario realizar de manera consciente estas asignaciones y apoyarse de la operación de modelado o **cast**.

(cast)

- **b=(int)a;** `//Antes de realizar la asignación a "b" se moldea a "a" como entero.`
- **b=(unsigned int)k;** `//Antes de realizar la asignación a "b" se moldea a "k" como entero sin signo.`
- **c=(float)g;** `//Antes de realizar la asignación a "c" se moldea a "g" como flotante.`
- **c=(char)j;** `//Antes de realizar la asignación a "c" se moldea a "j" como char.`

Expresiones condicionales; operadores relacionales y lógicos

- Si se tienen expresiones entre operadores de relación, se puede decir que dichas expresiones son condicionales, las cuales a su vez pueden estar unidas mediante operadores lógicos para obtener una condición múltiple.

- Los operadores de relación tienen la misma precedencia y esta por debajo de los operadores aritméticos.

- Las expresiones condicionales son evaluadas de izquierda a derecha y si se encuentran operadores lógicos son tratados en ese orden.

Operadores Relacionales

Operador	Nombre	Descripción
==	Igual a	if (a==‘s’)
!=	Diferente de	if (a!=null)
>	Mayor que	if (a>0.5)
<	Menor que	if (a<2l)
>=	Mayor o igual que	if (a>=2f)
<=	Menor o igual que	if (a<=3)

Operadores Lógicos

Operador	Nombre	Descripción
&&	Y (AND)	if ((a>3) && (a<9))
||	O (OR)	if ((a==2) || (a==3))
!	NEGADO (NOT)	if (!(a==3)) es igual a if (a!=3)

- Importante: Cuando se evaluá una condicional, los valores que se obtienen son:
- Si es FALSA se obtiene cero.
- Si es VERDADERA se obtiene algo diferente de cero.

Ejemplo de condicionales; operadores relacionales y lógicos

```c
#include<stdio.h>
#include<math.h>
int main(void)
{
        const float pi=3.1416;
        float distancia,x1=10,y1=11,x2=20,y2=16;

        //Cálculo de la distancia entre dos puntos
        distancia=sqrt(pow((x2-x1),2)+pow((y2-y1),2));

        if(distancia>0&&x2-x1<0&&y2-y1<0)
                printf("\nEl punto P2(%.2f,%.2f) esta debajo de
P1(%.2f,%.2f)",x2,y2,x1,y1);
        if(distancia>0&&x2-x1>0&&y2-y1>0)
                printf("\nEl punto P2(%.2f,%.2f) esta por encima de
P1(%.2f,%.2f)",x2,y2,x1,y1);
        if(distancia==0)
                printf("\nEl punto P2 y P1 están en las mismas
coordenadas(%.2f,%.2f)",x2,y2);
        return 0;
}
```

Operaciones para manejo de bits

- El lenguaje C proporciona seis operadores para manejo de bits; sólo pueden ser aplicados a variables enteras, esto es, char, short, int y long, con o sin signo.
  - No se debe confundir el operador & con el operador &&, & es el operador Y sobre bits, && es el operador lógico Y. Similarmente los operadores | y ||.
  - El operador unario ~ sólo requiere un argumento a la derecha del operador.
  - Los operadores de desplazamiento, >> y <<, mueven todos los bits en una posición hacia la derecha o la izquierda un determinado número de posiciones.
  - Como los operadores desplazan bits en un sentido, la computadora trae ceros en el otro extremo. Se debe recordar que un desplazamiento no es una rotación: los bits desplazados en un extremo no vuelven al otro. Se pierden y los ceros traídos los reemplazan.

Operadores de Bits

Operador	Nombre	Descripción
<<	Corrimiento a la izquierda	b=a>>2;
>>	Corrimiento a la derecha	b=a<<3;
&	Y (AND) entre bits	c=a&128;
|	O (OR) entre bits	c=a|0x0a;
~	Complemento A1	c=~a;
^	O exclusivo (XOR)	c=^a;

Operaciones del manejo de bits

- El operador AND de bits (&) a menudo es usado para enmascarar algún conjunto de bits.
- El operador OR (|) es empleado para encender bits.
- El operador OR exclusivo (^) pone uno en cada posición donde sus operandos tienen bits diferentes.
- El operador unario (~) realiza el complemento A1; esto es convierte cada bit 1 en un bit 0 y viceversa.

- Los operadores << y >> realizan corrimientos a nivel de bits, ya sea a la izquierda o derecha.

- Una aplicación que tienen los operadores de desplazamiento de bits es para realizar multiplicaciones y divisiones rápidas con enteros. Como se ve en la siguiente tabla, donde un desplazamiento a la izquierda es multiplicar por 2 y uno a la derecha dividir por 2.

- Los desplazamientos son mucho más rápidos que la multiplicación actual (*) o la división (/) por dos. Por lo tanto, si se quieren multiplicaciones o divisiones rápidas por 2 se usan desplazamientos.

- Multiplicaciones y divisiones por 2 al realizar corrimientos (<< y >>).

char x	Ejecución	Valor de x
x = 7;	0 0 0 0 0 1 1 1	7
x << 1;	0 0 0 0 1 1 1 0	14
x << 3;	0 1 1 1 0 0 0 0	112
x << 2;	1 1 0 0 0 0 0 0	192
x >> 1;	0 1 1 0 0 0 0 0	96
x >> 2;	0 0 0 1 1 0 0 0	24

Ejemplo: Operaciones del manejo de bits

```c
#include<stdio.h>
int main(void)
{
        int byte=0xFF;
        printf("\nEl valor de byte es: %4X H",byte);
        byte&=0x00;
        printf("\nEl valor de byte es: %4X H",byte);
        byte|=0xFF;
        printf("\nEl valor de byte es: %4X H",byte);
        byte>>=1;
        printf("\nEl valor de byte es: %4X H",byte);
        byte<<=2;
        printf("\nEl valor de byte es: %4X H",byte);
        return 0;
}
```

Operadores de Asignación para bits

Operador	Abreviado	No Abreviado
<<=	n<<=2;	n=n<<2;
>>=	n>>=2;	n=n>>2;
&=	n&=0x0a;	n=n&0x0a;
|=	n|=7;	n=n|7;
^=	n^=0x03;	n=n^0x03;
=	n=0x7f;	n=0x7f;

Nota:

0x7f, 0x0a, 0x03 son números hexadecimales.

La librería estándar <math.h>

- **math.h** es un archivo de cabecera de la biblioteca estándar del lenguaje de programación C diseñado para operaciones matemáticas básicas.

- Muchas de sus funciones incluyen el uso de números en coma flotante.
  - Una nota importante: si se está programando en C/C++ bajo Linux, y se utiliza uno de los compiladores nativos de este sistema operativo (gcc o g++), es necesario incluir, al compilar, la opción -lm, dado que, de lo contrario, el compilador generará un error.

- La biblioteca math.h contiene la definición de muchas funciones matemáticas útiles. El siguiente es un listado breve de algunas funciones.

`ceil(x)` Redondea al entero más pequeño no menor que x.
`cos(x)` Coseno de x.
`exp(x)` e^x.
`fabs(x)` Valor absoluto de x.
`floor(x)` Redondea al entero más grande no mayor que x.
`log(x)` Logaritmo natural de x.
`log10(x)` Logaritmo base 10 de x.

`pow(x,y)` x^y.
`sin(x)` Seno de x.
`sqrt(x)` Raíz cuadrada de x.
`tan(x)` Tangente de x.
`tanh(x)` Tangente hiperbólica de x.
`cosh(x)` Coseno hiperbólico de x.
`sinh(x)` Seno hiperbólico de x.
`fmod(x)` Resto del punto flotante de x.

***Todas las funciones en las que participan ángulos toman y devuelven radianes.**
***Todas operan con parámetros de tipo double (también se soportan flotantes) y devuelven resultados de tipo double (pueden guardarse como flotantes).**

Constantes <math.h>

- La biblioteca de matemáticas define varias constantes. Siempre es aconsejable usar estas definiciones.
  - **M_E** La base de los logaritmos naturales e.
  - **M_LOG2E** El logaritmo de e de base 2.
  - **M_LOG10E** El logaritmo de e de base 10.
  - **M_LN2** El logaritmo natural de 2.
  - **M_LN10** El logaritmo natural de 10.
  - **M_PI** pi
  - **M_PI_2** pi/2
  - **M_PI_4** pi/4
  - **M_1_PI** 1/pi
  - **M_2_PI** 2/pi
  - **M_2_SQRTPI** 2/sqrt(pi)
  - **M_SQRT2** La raíz cuadrada positiva de 2
  - **M_SQRT1_2** La raíz cuadrada positiva de 1/2

Ejemplo empleando <math.h>

```c
#include<stdio.h>
#include<math.h>
int main(void)
{
	const float pi=3.1416;
	float distancia,x1=1.5,y1=-10,x2=2.5,y2=20;
	//Mostrando la diferencia de exactitud entre pi y M_PI
	printf("\nEl seno de pi es: %f",sin(pi));
	printf("\nEl seno de M_PI es: %f",sin(M_PI));

	//Cálculo de la distancia entre dos puntos
	distancia=sqrt(pow((x2-x1),2)+pow((y2-y1),2));
	printf("\nLa distancia entre P2(%.2f,%.2f) y P1(%.2f,%.2f)
es: %.4f",x2,y2,x1,y1,distancia);
	return 0;
}
```

Su compilación en Linux sería: gcc programa.c –lm –o programa

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería de la Computación

Academia de Ciencias de la Computación

Autor: M. en C. Edgardo Adrián Franco Martínez

Algoritmia y programación estructurada

Unidad I "Conceptos básicos y herramientas de programación"

1.6 Flujo de ejecución

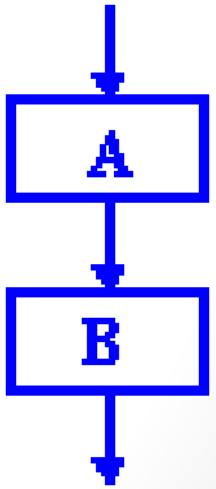
Contenido

- Estructuras secuenciales
- Estructuras selectivas
  - if
  - If else
  - If -else if
  - switch-case
  - operador ?
- Estructuras iterativas
  - while
  - do-while
  - For
- Sentencias de salto
  - return
  - goto
  - break
  - continue

Estructuras secuenciales

- La secuencia de pasos es la más fácil de las estructuras de control, ya que en Lenguaje C basta con escribir los pasos en orden descendente y las instrucciones se ejecutarán de manera secuencial.

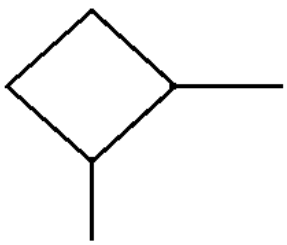
```
//01 Declarar tres variables enteras
 int x, y, z;
 //02 Igualar las tres variables enteras con VALOR
 x=y=z=VALOR;
 //03 Ejecutar la función modulo3D
 mod3=modulo3D(x,y,z);
 //04 Mostrar en la salida estándar el resultado
 printf("\nEl módulo es: %lf",mod3);
 //05 Salir del programa
 return(0);
```

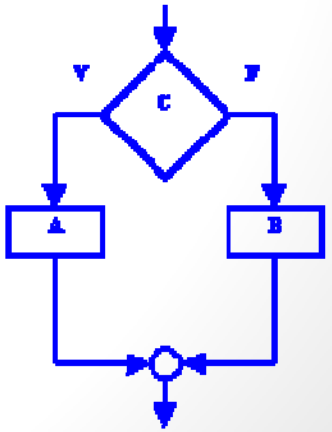


Estructuras de selección

- Las estructuras de selección (o bifurcación) se utilizan para elegir entre diversos cursos de acción.

- En C hay tres tipos de estructuras de selección: **if** (selección simple), **if…else** (selección doble) y **switch** (selección múltiple)





Selección (if)

```
if (expresión)
  sentencia;
```

Nota: una expresión en C es todo aquello que regresa un valor. Como por ejemplo una condición lógica, operaciones aritméticas, llamadas a funciones, una variable, una constante (numérica, carácter, etc.).

```
if (expresión)
 {
  sentencia1;
  sentencia2;
 }
```

Selección (if – else)

```
if (expresión)
  sentencia;
else
  sentencia;
```

```
if (expresión)
 {
  sentencia1;
  sentencia2;
 }
else
 {
   sentencia1;
   sentencia2;
 }
```

Selección (if – else if)

```
if (expresión)
  sentencia;
else if (expresión)
  sentencia;
…
else
  sentencia;
```

```
if (expresión)
 {
  sentencia1;
  sentencia2;
 }
else if
 {
   sentencia1;
   sentencia2;
 }
…
else
 {
   sentencia1;
   sentencia2;
 }
```

Selección (switch-case)

```
switch(expresión)
 {
   case 1:  sentencias;
            break;
   case 2:  sentencias;
            break;
    :
   case n:  sentencias;
            break;
   default: sentencias_default;
            break;
 }
```

Selección (operador ?)

(expresión)? sentencia1 : sentencia2;

expresión? sentencia1 : sentencia2;

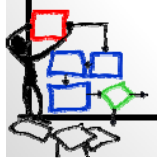
Se ejecuta:

sentencia1 si expresión = verdadero

sentencia2 si expresión = falso.

Es un operador ternario y puede utilizarse para asignar variables:

Var = (expresión)? sentencia1:sentencia2;



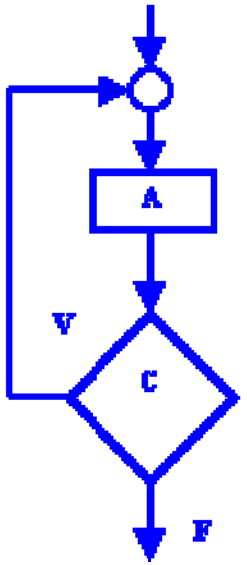
Selección (ejemplos)

```
if (calificacion>=60)
{
        printf ("Alumno Aprobado");
}
else
{
        printf ("Alumno Reprobado");
}
```

```
if (calificacion>=90)
{
        printf ("Alumno de muy buena calificación");
}
else if (calificacion>=70&& calificacion<90)
{
        printf ("Alumno regular");
}
else if (calificacion>=60&& calificacion<70)
{
        printf ("Alumno");
}
else
{
        printf ("No es alumno");
}
```

Estructuras iterativas

- Estas estructuras permiten repetir una serie de veces la ejecución de unas líneas de código.
- Esta iteración se realiza o bien un número determinado de veces o bien hasta que se cumpla una determinada condición de tipo lógico o aritmético.

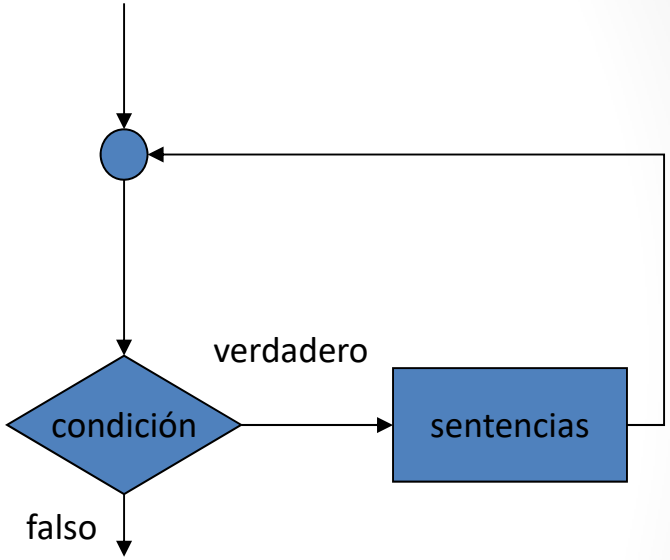


- En C hay 3 tipos de instrucciones de repetición: **while**, **do**…**while** y **for**.

Estructura iterativa (while)

- La sentencia while permite repetir un bloque de instrucciones.
- La sintaxis del ciclo while es:

```
while(condición)
{
   sentencia o bloque;
};
```

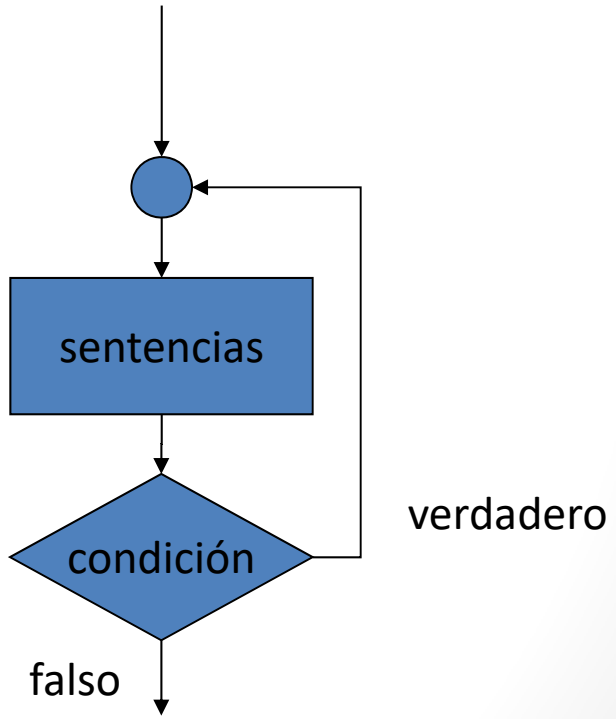


- Si la condición se cumple se ejecutan las sentencias del bloque y se regresa el flujo de control a evaluar nuevamente la condición. El proceso se repite hasta que la condición sea falsa.
- El ciclo puede ejecutarse 0 veces si la condición no se cumple.

Estructura iterativa (do-while)

- El ciclo do-while es similar al ciclo while excepto que la condición se realiza al final del ciclo, esto fuerza a que se ejecute por lo menos una vez.

```
do{
        sentencias;
}while(condición);
```



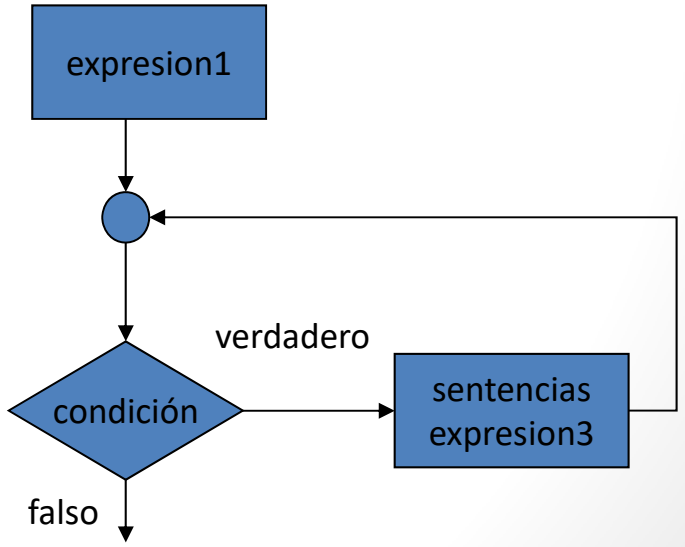
Modelado de la iteración (For)

- La sentencia for permite definir fácilmente ciclos controlados por contador.

```
for(expresion1; expresion2; expresion3)
{
      sentencias;
}
```

- Esta es equivalente a la siguiente sentencia while:

```
expresion1;
while(expresion2)
{
  sentencias;
  expresion3;
}
```



expresion1 = sentencia de iniciación
expresion2 = condición de terminación
expresion3 = sentencia de incremento

Sentencias de salto de C

- En C hay sentencias para evitar la ejecución estrictamente secuencial del programa. En general (sentencias ***goto, break*** **y** ***continue***) no deben utilizarse para elaborar programas estructurados, aunque pueden ser útiles si se justifica en forma apropiada su aplicación en un código.
- **return <expresión>;**
  - Se usa para devolver el control del flujo de ejecución desde una función, siendo <expresión> el valor (dato) retornado por ella.
- **goto <etiqueta>; …. <etiqueta>: <acciones>**
  - Permite efectuar un salto incondicional hasta otro punto del programa, indicado por una etiqueta.

```c
#include<stdio.h>
int main(void)
{
        printf("\nEl programa ha iniciado");
        printf("\nInstrucción 1");
        printf("\nInstrucción 2");
        goto se_me_antoja;
        printf("\nInstrucción 3");
        printf("\nInstrucción 4");
se_me_antoja:
        printf("\nInstrucción 5");
}
```

- **break;**
  - Se usa para romper el flujo en un switch-case o terminar sentencias iterativas (for, while, do-while).
- **continue;**
  - Se puede usar en un bloque iterativo (for, while, do-while) para forzar una nueva iteración del ciclo ignorando las sentencias que están a partir de **continue** y hasta el fin del ciclo.

```c
#include<stdio.h>
int main(void)
{
        short i;
        for(i=0;i<100;i++)
        {
                printf("\n%d",i);
                break;
                printf("\nDespues del break");
        }
}
```

```c
#include<stdio.h>
int main(void)
{
        short i;
        for(i=0;i<100;i++)
        {
                printf("\n%d",i);
                continue;
                printf("\nDespues del continue");
        }
}
```