Facultad de Informática

PROGRAMACIÓN I
Sesiones teóricas

Departamento de Ciencias de la Computación
y Tecnologías de la Información

SESIONES TEÓRICAS

Metodología - Planificación

- Clases magistrales (aula): **2h cada semana**
- **Objetivo**: **conocer y comprender los conceptos teóricos** que forman el contenido de la asignatura Programación I
- **Método**: **exposición** de los **conceptos** integrantes de cada sesión teórica, indicados como lectura recomendada en la guía de la asignatura

SESIONES TEÓRICAS

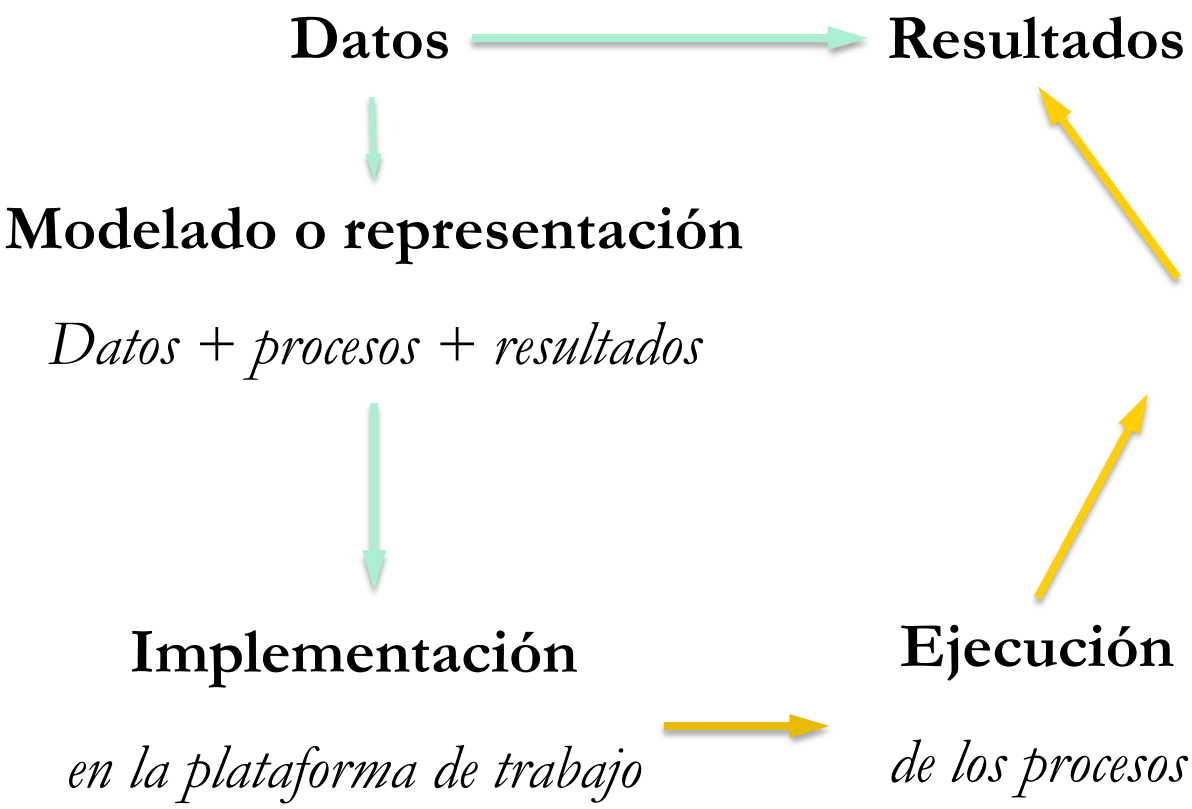
CONCEPTOS BÁSICOS

- **ALGORITMOS**
  - Representación de Algoritmos
- **PROGRAMAS**
  - Tipos de programas
- **LENGUAJES DE PROGRAMACIÓN**
  - Una visión histórica . Lenguaje C
  - Instrucciones más importantes
- **DESCRIPCIÓN DE LOS LENGUAJES**
  - Notación BNF. Diagramas de Conway
- **ESTRUCTURA. ELEMENTOS DE UN PROGRAMA**
  - Sentencias
  - Datos. Tipos de datos.
  - Sentencias E/S
  - Conversión de tipos
  - Introducción a los punteros
  - Introducción a las macros

CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

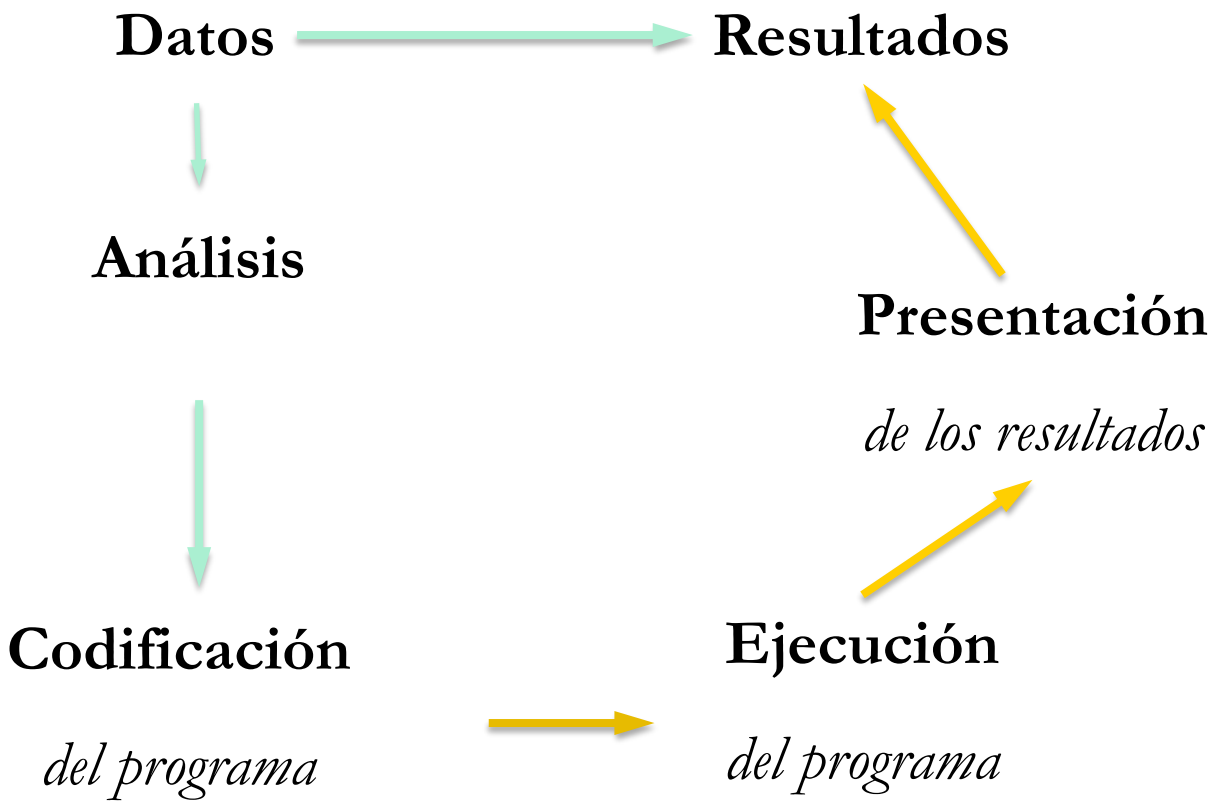
¿Cómo se resuelve un problema en general?



CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

¿Cómo se resuelve un problema en un computador?



CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

- Problemas que se tratan de resolver en Programación
  - Cómputo o cálculo
  - Gestión comercial
  - Control
  - Tratamiento de la señal
  - Lúdicos
  - Inteligencia Artificial
  - E-comercio
  - Etc.

- Problemas cuya solución puede ser llevada a cabo por un **COMPUTADOR**

  Herramienta que permite tratar de forma automática problemas de tratamiento de la información

CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

¿Cuál es el objetivo de la PROGRAMACIÓN?

CONSTRUIR PROGRAMAS

- **Programa**: **texto formado por un conjunto de sentencias que se desea que ejecute una máquina**
  - Los lenguajes de programación están diseñados de forma que sólo se requiere que los **programadores** indiquen sus intenciones **explícitamente**

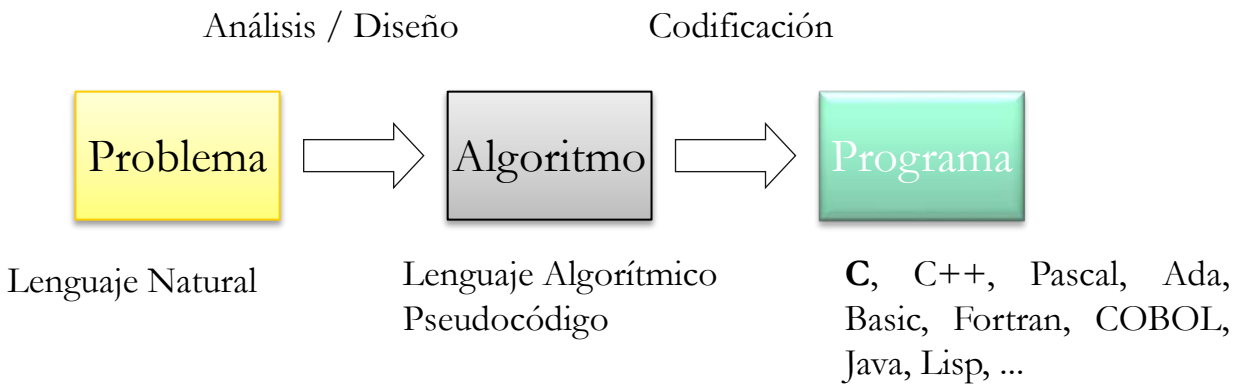
SÍ, PERO……..

¿Cómo transformar un problema en un programa ejecutable por un computador?

CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

- **Entre problema y programa hay algunos pasos importantes**



CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

- **Algoritmo**: conjunto **ordenado** y **finito** de pasos o **sentencias** que especifican, de forma precisa y sin ambigüedades, la **secuencia de operaciones** que se han de realizar para resolver un problema

Iremos calculando

factorial de n =

1* 2 * ...*i * ... n

para valores crecientes de i hasta n

```
i:=1;
factorial:=1;
Mientras Que i < n hacer
  i:= i+1;
  factorial:= factorial * i
FinMientrasQue;
```

Solución informal en *lenguaje natural*

Algoritmo en *lenguaje algorítmico*

CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

- **Programa**: codificación en cualquier lenguaje de programación específico de uno o varios algoritmos

```
i:=1;
factorial:=1;
Mientras Que i < n hacer
  i:= i+1;
  factorial:= factorial * i
FinMientrasQue;
```

Algoritmo en *lenguaje algorítmico*

```
i=1; factorial=1;
While (i < n){
  i= i+1;
  factorial=factorial * i;
}
```

Programa en *C*

CONCEPTOS BÁSICOS

ALGORITMOS. Conceptos

Solución informal en *lenguaje natural*	Algoritmo en *lenguaje algorítmico*	Programa en *lenguaje C*

Se va calculando
factorial de n=
1* 2 *...*i *...n
para valores crecientes de i
hasta llegar a n
	`i ← 1;`
	`fact ← 1;`
	`MientrasQue i<n hacer`
	`  i ← i+1;`
	`  fact ← fact * i`
	`FinMientrasQue;`
		`i=1;`
		`fact=1;`
		`while (i < n) {`
		`  i=i+1;`
		`  fact=fact*i;`
		`}`

CONCEPTOS BÁSICOS

ALGORITMOS. Propiedades

- Condiciones Necesarias
  - Finitud
  - No ambigüedad

- Propiedades deseables
  - Generalidad
  - Eficiencia
  - Independencia de la máquina

CONCEPTOS BÁSICOS

ALGORITMOS. Representación

- **Diagramas de flujo u ordinogramas**
- Diagramas N-S (Nassi-Schneiderman)
- **Pseudocódigo**
- Lenguaje natural
- Métodos formales matemáticos

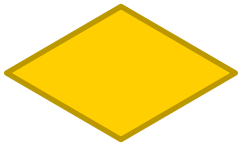
CONCEPTOS BÁSICOS

ALGORITMOS. Ordinograma

Terminal: Inicio o Fin

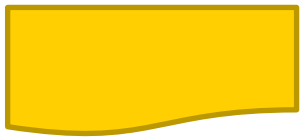
Flujo de control o dirección

Instrucción



Decisión

Entrada/Salida de datos

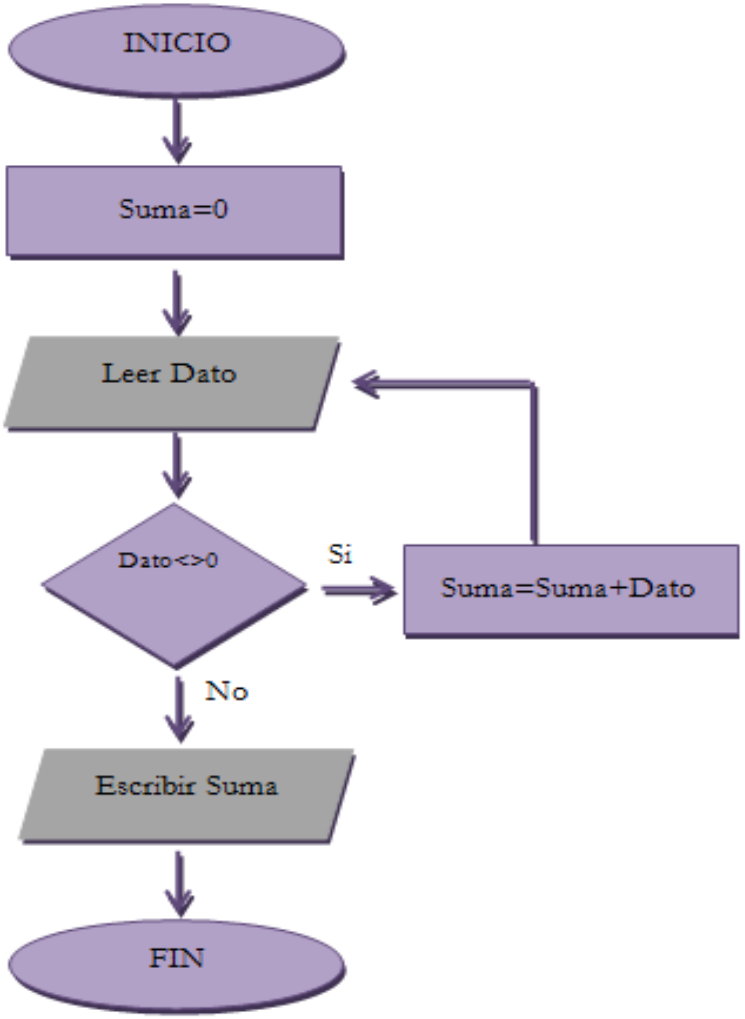


Salida por **impresora**

CONCEPTOS BÁSICOS

ALGORITMOS. Ordinograma

- Ejemplo: ordinograma para sumar un número variable de números



CONCEPTOS BÁSICOS

ALGORITMOS. Pseudocódigo

```
MIENTRAS no llegue al final del fichero
    Leer registro
    SI Edad > 67 años ENTONCES
        Imprimir Apellidos y nombre
    SINO
        Imprimir DNI y Profesión
    FIN SI
FIN MIENTRAS
```

CONCEPTOS BÁSICOS

PROGRAMAS. Tipos

- Programas de aplicación
- Editores
- Compiladores
- Traductores
- Montadores de enlace (*linkers*)
- Sistemas operativos
- Programas de servicio

CONCEPTOS BÁSICOS

PROGRAMAS. Sistema Operativo

- Sistema Operativo
  - Conjunto de programas que tienen como objetivo **facilitar** la utilización del computador
    - Acceso a los usuarios autorizados
    - Edición de ficheros de texto
    - Configuración y ejecución de programas
    - Seguridad y protección
    - etc.
  - **Gestión óptima** de la máquina
    - Gestión de memoria
    - Control de dispositivos periféricos
    - Acceso a ficheros
    - Asignación de recursos y ordenación de tareas
    - etc.

CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Tipos

- **Lenguaje máquina**
- **Lenguaje ensamblador**
- **Lenguajes de alto nivel**
  - Lenguajes compilados
  - Lenguajes interpretados
  - Lenguajes híbridos

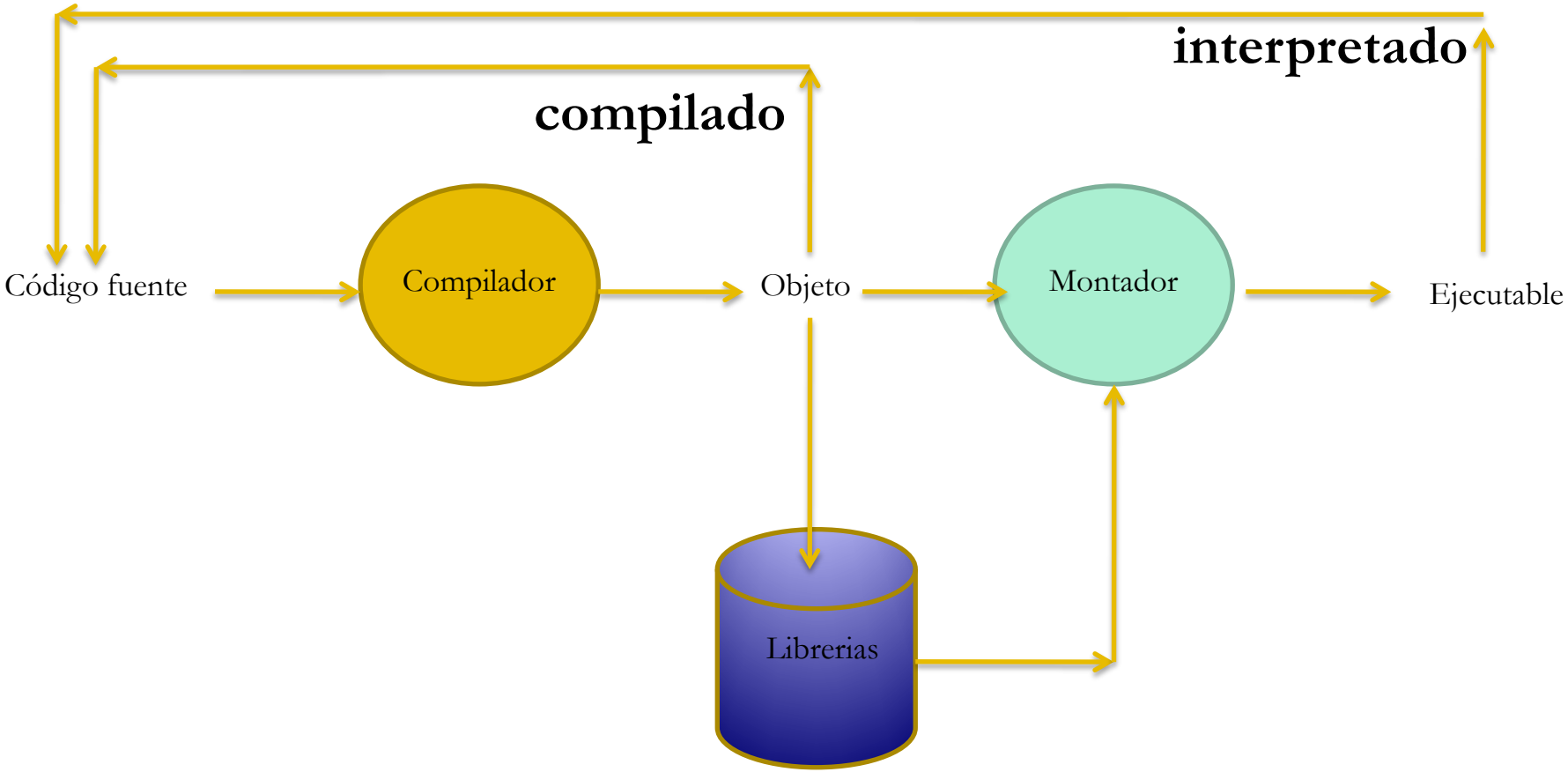
CONCEPTOS BÁSICOS

LENGUAJES DE PROG. Compilados/Interpretados

- Ejecución **interpretada** del programa fuente
  - Se iteran los siguientes pasos
    - **Obtención de la siguiente sentencia*** a ejecutar del código fuente
    - Análisis de la sentencia y **determinación de las acciones a ejecutar**
    - **Ejecución** de las correspondientes acciones

- Ejecución del **programa compilado**
  - Se realiza previamente la compilación del **programa fuente** en la versión equivalente del programa en lenguaje máquina. Esta versión se conoce como **programa objeto**
  - El **programa objeto** se une, eventualmente, con los subprogramas de biblioteca descritos en el programa fuente para obtener el **programa ejecutable**: esto lo realiza el montador de enlaces (*linker*)

CONCEPTOS BÁSICOS
LENGUAJES DE PROG. Edición/Compilación/Montaje



CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Propiedades

- Claridad
- Buena definición
- Independencia de la máquina
- Notación estándar
- Niveles de abstracción
- Permita programar sin errores
- Verificación de programas

CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Paradigmas

- Imperativa
  - C, Pascal, Cobol, Fortran, etc.
- Orientada a objetos
  - Java, C++, C#, .NET
- Funcional
  - LISP, Scheme, ML
- Lógica o declarativa
  - PROLOG
- Programación Concurrente
- Tiempo real

CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Ejemplos

- **FORTRAN** (1965): FORmula TRANslator. Calculo científico
- **COBOL** (1959). Idóneo para gestión: pocas estructuras de control, muchas capacidades de manejo de datos
- **ALGOL** (1960): primer intento de formalizar la resolución de problemas. Programas estructurados con tipos de datos
- **Lisp** (1960): Actualmente en uso. Estructura de datos simple (la lista). Tratamiento simbólico de la información. Su éxito se debe a que soporta más estrategias de descomposición modular que cualquier otro lenguaje
- **BASIC** (1965): Beginners All Purpose Symbolic Language. Lenguaje interpretado, sin énfasis en tipos de datos
- **PASCAL** (1971): Desciende de ALGOL. Paradigma de la programación estructurada

CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Ejemplos

- **C** (1971): desarrollado por Dennis Ritchie de AT&T Bell Laboratories para ofrecer un lenguaje de alto nivel para programar UNIX.
- **Lenguajes orientados a objetos** (C++ (1983), Eiffel (1986), C# (2000),…): a la hora de organizar el software se focalizan en las abstracciones de datos en lugar de en la funcionalidad.
- **HTML** (1991): lenguaje de marcado para el desarrollo de páginas web, utilizado para dar formato y estructura a la información en forma de texto e imágenes.
- **Phyton** (1991): lenguaje de propósito general centrado en la legibilidad del código que soporta múltiples paradigmas (imperativo, funcional, orientado a objetos, etc.).
- **JAVA** (1995): lenguaje orientado a objetos que, teóricamente, se podría ejecutar en cualquier máquina, gracias al concepto de máquina virtual (filosofía *write once, run anywhere*).
- **Dart** (2011)**:** lenguaje de programación web en código abierto desarrollado por *Google®,* con el objetivo de sustituir a javascript en el desarrollo web.

Origen del Lenguaje C

- C es un producto derivado del UNIX, desarrollado en los Laboratorios Bell por Ken Thompson, Dennis Ritchie y otros.
- Thompson diseñó un pequeño lenguaje llamado B.
- B a su vez se basó en BCPL, un lenguaje de programación de sistemas desarrollado a mediados de los años sesenta. En 1971, Ritchie comenzó a desarrollar una versión extendida de B. Le llamó NB ("Nuevo B") al principio. A medida que el lenguaje comenzó a distanciarse más de B, cambió su nombre a C.
- El lenguaje era lo suficientemente estable en 1973 para que UNIX se pudiera reescribir en C.

Estandarización del lenguaje C

- **K&R C**
  - El lenguaje C fué descrito y analizado en el libro de Kernighan and Ritchie, *The C Programming Language (1978)*. De facto fué (se asumió) como un estandar.

- **C89/C90**
  - ANSI estandar X3.159-1989 (completado en 1988; formalmente aprovado en Diciembre de 1989)
  - International standard ISO/IEC 9899:1990.

- **C99**
  - International standard ISO/IEC 9899:1999.

Propiedades del lenguaje C

- Bajo nivel.
- Pequeño.
- Sintaxis abierta.
- Eficiencia.
- Portabilidad.
- Poder (asociado al acceso y control de la máquina).
- Flexibilidad.
- Biblioteca estándar.
- Integración con UNIX.

Lenguajes basados en C

C ++, incluye todas las funciones de C, pero agrega clases y otras características para admitir la programación orientada a objetos

Java, se basa en C ++ y, por lo tanto, hereda muchas características de C. Es un lenguaje de programación y una plataforma informática de desarrollo comercializada por primera vez en 1995 por Sun Microsystems

C#, es un lenguaje más reciente derivado de C ++ y Java, es un lenguaje desarrollado y estandarizado por Microsoft como parte de su plataforma .NET

Perl, es un lenguaje diseñado por Larry Wall en 1987 ha adoptado muchas de las características de C.

Elementos

CONCEPTOS BÁSICOS

LENGUAJES DE PROGRAMACIÓN. Sentencias

- **ASIGNACIÓN**
  - Evaluar una expresión y almacenar el resultado en una variable
    - a=3+4;
- **ENTRADA/SALIDA**
  - Transferencia de información con el exterior a través de dispositivos de entrada (teclado, ficheros, etc.) y salida (monitor, impresora, etc.)
    - printf, sacanf, fputc…
- **CONTROL DE FLUJO**
  - **Selección**
    - **if , if-else, switch.**
  - **Repetición**
    - **while, do-while, for**
- **LLAMADA Y RETORNO DE SUBRUTINAS**
  - Dividir el programa en pequeñas unidades que se ejecutarán cuando convenga

CONCEPTOS BÁSICOS

DESCRIPCIÓN. Elementos

- Elementos que definen un lenguaje de programación
  - **Alfabeto**: conjunto de símbolos válidos para construir textos en el lenguaje
    - Palabras clave = {IF, ELSE, WHILE, SWITCH …}
    - Caracteres = {‘a’,..’z’,’A’..’Z’,’#’,’?’,……. }
    - Dígitos = {0,..,9}
    - Otros símbolos = {‘.’ , ’,’ , ‘;’ ,…}
  - **Léxico:** vocabulario formado a partir del alfabeto
    - IF, ELSE, WHILE
  - **Sintaxis**: conjunto de reglas gramaticales que permiten derivar *frases* correctas en el lenguaje
    - Notaciones para expresar las reglas:
      - **Backus-Naur**
      - **Grafo Sintáctico**
  - **Semántica:** conjunto de reglas que permiten derivar *frases* (sentencias) con sentido o significado en el lenguaje

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF

- Creada por J. **Backus** y P. **Naur** para describir la sintaxis del lenguaje ALGOL 60
- Para describir el lenguaje se utilizan **reglas** que se construyen con tres tipos de símbolos:
  - **Metasímbolos**: propios de la notación BNF
  - **Símbolos terminales**: son los que se usarán en el texto del programa tal y como aparecen en la regla (sin comillas)
  - **Símbolos no terminales**: resto de símbolos, se definen utilizando combinaciones de símbolos terminales, no terminales y metasímbolos

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF

Metasímbolo	Significado
::=	Se define como
|	Opción alternativa
(a|b)	a ó b pero no ambos
[palabra]	0 ó 1 vez palabra
{palabra}	0 ó más veces palabra
'xyz'	Símbolo terminal
.	Final de la regla

- Definición de un dígito con 10 reglas
  - digito ::= '0'.
  - digito ::= '1'.
  - …
  - digito ::= '9'.
- Definición de un dígito con una única regla
  - digito ::= '0' | '1' | '2' | '3' | '4' | ……| '9'.

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF

Metasímbolo	Significado
::=	Se define como
|	Opción alternativa
(a|b)	a ó b pero no ambos
[palabra]	0 ó 1 vez palabra
{palabra}	0 ó más veces palabra
'xyz'	Símbolo terminal
.	Final de la regla

- Definición de un número entero
  - entero_sin_signo ::= digito {digito}.
  - signo ::= '+' | '-'.
  - entero ::= [signo] entero_sin_signo.

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF

Metasímbolo	Significado
::=	Se define como
|	Opción alternativa
(a|b)	a ó b pero no ambos
[palabra]	0 ó 1 vez palabra
{palabra}	0 ó más veces palabra
'xyz'	Símbolo terminal
.	Final de la regla

identificador ::= letra { [guion_bajo] (letra | digito)} .

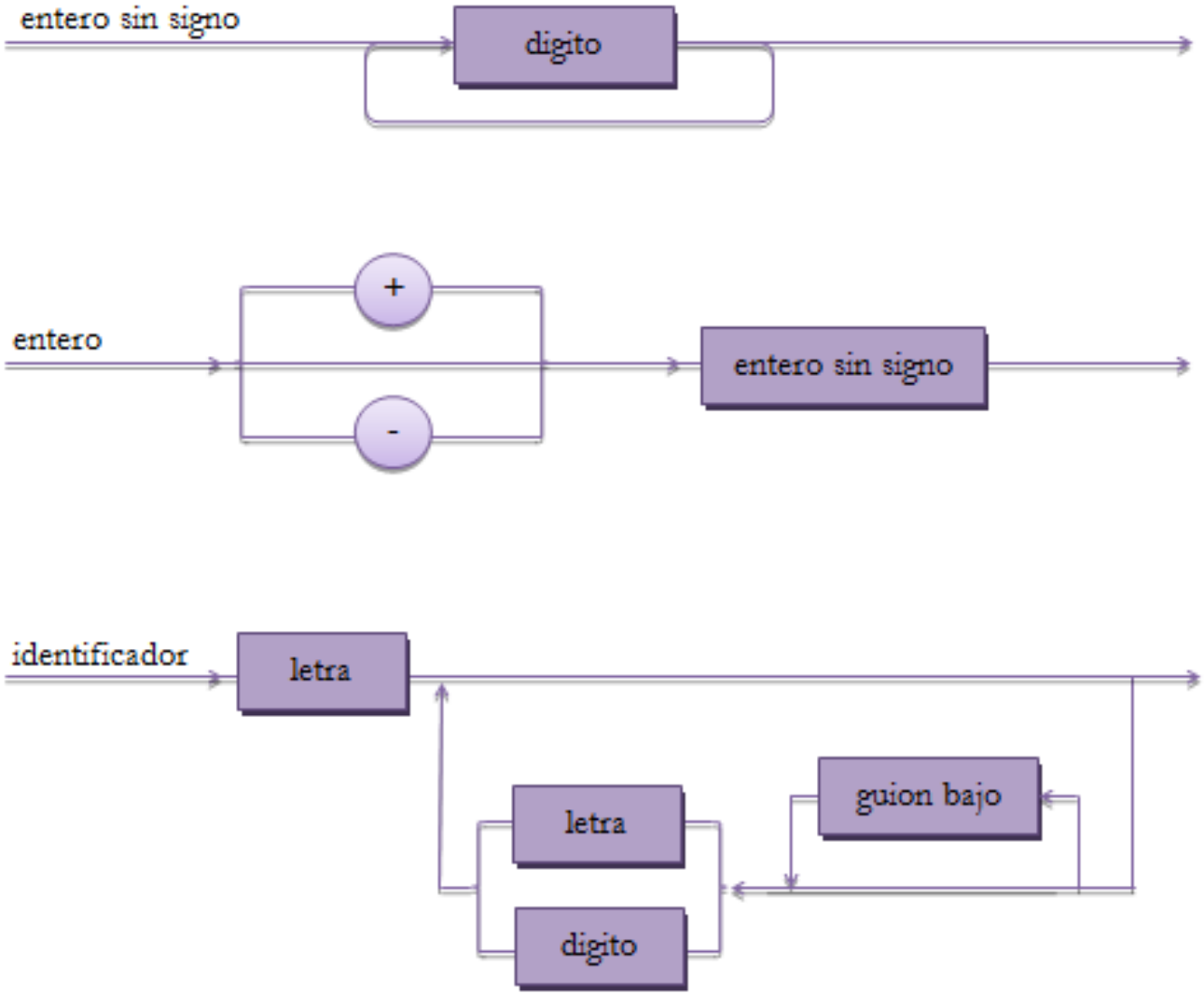
CONCEPTOS BÁSICOS

LENGUAJES. Diagramas de Conway

- **Herramienta gráfica** para describir la sintaxis de un lenguaje, se utilizan también símbolos terminales y no terminales:
  - **Símbolos terminales**: se representan por un círculo o rectángulo redondeado
  - **Símbolos no terminales**: se representan mediante un rectángulo
  - Ambos símbolos se enlazan con **flechas** que indican el orden en que se debe realizar la interpretación de los diagramas

CONCEPTOS BÁSICOS

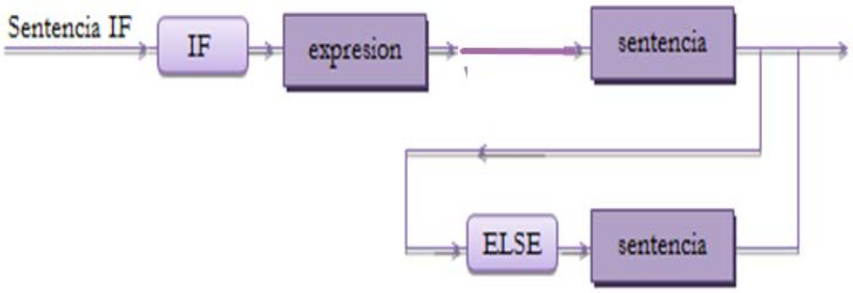
LENGUAJES. Diagramas de Conway



CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF/Diagramas de Conway

Conway



BNF

SentenciaIF ::=

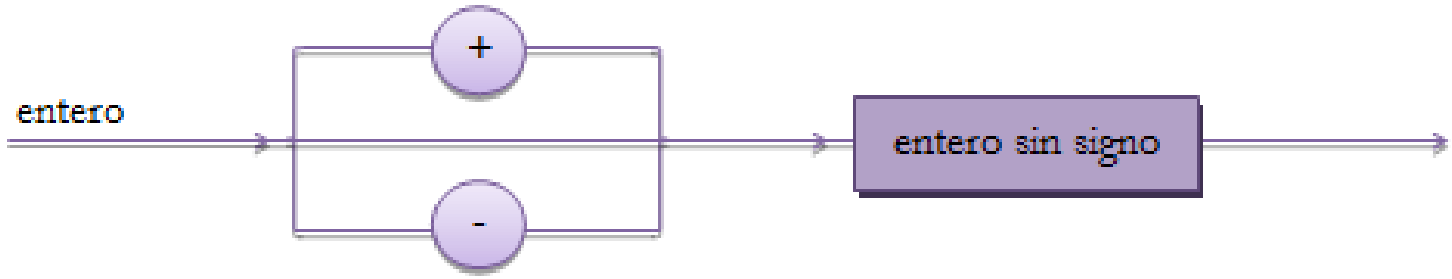
'IF' (expresión_booleana) sentencia

ELSE sentencia.

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF/Diagramas de Conway

Conway



BNF

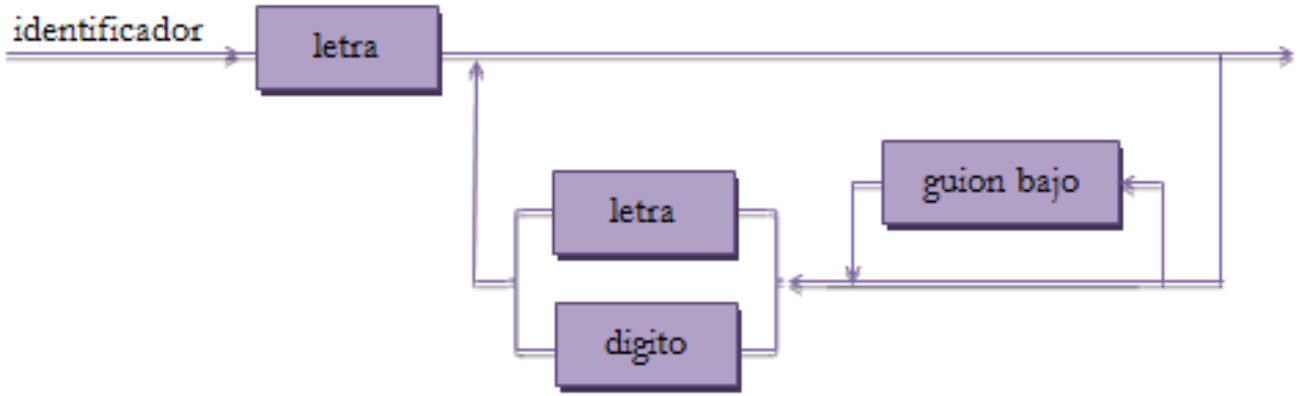
signo ::= ‘+’ | ‘-’.

entero ::= [signo] entero_sin_signo.

CONCEPTOS BÁSICOS

LENGUAJES. Notación BNF/Diagramas de Conway

Conway



BNF

identificador ::= letra { [guion_bajo] (letra | digito)}

CONCEPTOS BÁSICOS

Notación BNF/Diagramas de Conway

Conway

BNF

entero_sin_signo ::= digito {digito}.

CONCEPTOS BÁSICOS

Ejemplo primer PROGRAMA

```
#include <stdio.h>

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

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA

Un **programa** se crea utilizando un **alfabeto de símbolos** (léxico) para construir:

- identificadores
- etiquetas
- directivas remotas, de interface y de implementación
- constantes
- números sin signo, números reales
- cadenas de caracteres (*strings*) y variables.

CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Símbolos predefinidos

- **Símbolos predefinidos**: conjunto de símbolos que constituyen el **alfabeto** del lenguaje
  - En el caso de C será el juego de caracteres **ASCII**

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	¿	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Símbolos predefinidos

- **Tabla ASCII de 7 bits**:
  - Cada símbolo una posición
  - Número entero asociado a cada carácter
  - Los códigos 0-31 y 127 no son imprimibles
  - La diferencia entre el código ASCII de una letra mayúscula y su correspondiente minúscula es de 32

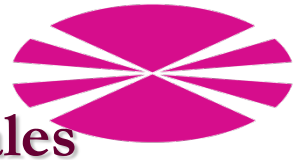
CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Símbolos predefinidos

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	np	cr	so	si	dle	dc1	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	"	#	$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	¿	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Símbolos especiales



- **Símbolos especiales:** símbolos con un **significado especial** dentro del lenguaje (C)que se usan para representar delimitadores, operadores y otros elementos sintácticos

special-symbol = '+' | '-' | '*' | '/' | '=' | '<' | '>' | '{' | '}'
| '&' | ',' | ':' | ';' | '^' | '(' | ')' | '['
| '' | '<=' | '>=' | '==' | '..' | '!=' |
| word-symbol .

CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Palabras reservadas.

- Las **palabras reservadas** (***keywords)*** se utilizan para especificar nombres de sentencias, funciones de librería, tipos de datos, operadores, etc.
  - **No se pueden redefinir** para usarlas como identificadores

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

* palabras reservadas a partir del compilador C99

CONCEPTOS BÁSICOS

ELEM. DE UN PROGRAMA. Identificadores

- **Identificadores:** son nombres que pueden utilizarse para **denominar** una **función**, una **constante**, una **variable**, un **tipo**, una **función**, una interfaz

  identifier = letter { [underscore] (letter | digit) } .
  underscore = '_' .

- **el nombre** debe ser **apropiado:** que tenga un significado adecuado al uso destinado para el identificador

  - Ejemplos: X, Edad, readinteger, WG2, Device_DriverIdentifier, etc.,

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA. Constantes

- **Constantes:** identificadores que se corresponden a un valor constante que **no varía** durante la ejecución del programa

  constant-definition = identifier '=' constant-expression .

  - **Ejemplos**:

```
ANIO = 1998;
mes = “Enero”;
PI = 3.1416;
unidad = 1.0;
tercio = unidad/3.0;
limite = 43;
blank  = ' ';
pi_bis  = 4 * arctan(1);
```

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA

- **Números**: en C pueden ser **enteros**, **reales** o **complejos**
  - **Ejemplos**: **1E10, +1, -3, -0.04, 5e-10, 86.44E+6**
- **Las secuencias de caracteres:** se especifican entre **comillas “ ”**

  “Hola mundo, otra vez”
- Si es solo **un carácter** usamos la **comilla sencilla** ´ , ´A´
  - Para especificar un **carácter constante** es necesario escribir cualquier **carácter ASCII imprimible** entre apóstrofes
  - Las cadenas pueden ser de **longitud fija** o **variable**

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA. Variables

- Los valores de una variable pueden **cambiar** durante la ejecución de un programa, **permaneciendo dentro del rango** establecido para su tipo.

- Se define como:

  tipo_de_datos nombre_variable;

  tipo_de_datos var1, var2, …varN;

- **Ejemplos**:

  - float precio ;
  - int i, j k ;

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA. Variables

- Cuando se declara una variable:
  - Se **presenta el identificador**, antes no existía
  - Se **reserva la memoria** necesaria para el tipo de datos de la variable
  - Se **asocia el identificador con la dirección** de memoria donde comienza
  - Se asocia el identificador con una "cierta" **interpretación de memoria**

- **Ejemplos**:

```
char letra;
char letra=´a´;
int edad;
float precio;
```

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA. Variables

- **Variables: no adoptan valores** hasta que el programa se los asigna, por ello es fundamental

INICIALIZARLAS

- En la declaración

  float precio=123.34;

- Mediante entrada por teclado

  scanf (“%f ”, &precio);

- Llamando a una función

  iniciar (precio);

- **En una sentencia de asignación**

  precio = 1150.25; (variable=valor)

**Los tipos de variable y valor han de ser los mismos (o compatibles).*

CONCEPTOS BÁSICOS

ELEMENTOS DE UN PROGRAMA. Comentarios

- **Comentarios:** cualquier **secuencia** de caracteres y separación de líneas que **no contengan ni } ni *)**

  - Un comentario comienza por /* y terminar con */
  - Los comentarios pueden intercalarse en **cualquier lugar** de un programa
  - Sirven para **DOCUMENTAR** los programas, no ejecutan ninguna acción
  - *En compiladores de C99 y posteriores se admite dos barras para indicar comentario:*
  - // estas barras indican un comentario

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

- En C todo **dato** está **asociado** a un **tipo de datos** y debe declararse antes de utilizarse
- Un tipo de datos está **representado por un identificador**
- Un tipo de datos determina el **rango de valores** que puede tomar un elemento y las **operaciones** que se pueden realizar sobre y con él

Tipos de Datos /Operadores

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

Clasificación de tipos de datos

- **Según sean sus constituyentes**
  - Simples
  - Compuestos
    - Estructurados
    - No estructurados
- **Según quien los haya definido**
  - Predeterminados
  - Definidos por el usuario
- **Según la relación entre los valores de referencia**
  - Ordinales
  - No ordinales
- **Según la denominación**
  - Con nombre
  - Sin nombre (Anónimos)

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

Clasificación de tipos de datos

		Tipo C	Tipo de datos
TIPOS SIMPLES		INT (varios)	Números enteros
		CHAR	Caracteres
	Ordinales	Bool *	Valores lógicos (true/false)
		ENUM	Enumeración
	No ordinales	FLOAT, DOUBLE (varios) COMPLEX *	Números reales Números complejos
TIPOS ESTRUCTURADOS		ARRAY *	Vectores/Matrices
		STRUCT	Registros
		FILE	Ficheros

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

Tipos de datos SIMPLES

- Predeterminados
  - Numéricos (**INT**, **FLOAT, DOUBLE…**)
    - 3, +9, -125, 3.14 , 976.251, 7.0, 0.0, 2.3e2, 10.0E-1
  - Caracteres (**CHAR**)
    - 'A', '.', ' ', 'Z', '0', '?'
  - Punteros *
- Definidos por el usuario
  - Enumerados (**ENUM**)

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES

- **ARITMÉTICOS**: proporcionan una ecuación para calcular un valor.
- **RELACIONALES**: comparan dos expresiones de tipo ordinal, real, cadena de caracteres o conjunto, obteniendo un resultado de tipo lógico (true, false).
- **LÓGICOS**: evalúan una o más expresiones lógicas, obteniendo como resultado otro valor lógico (true, false).

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

DATOS en C

char

char, signed char, unsigned char

int

int, signed / unsigned int

short, signed / unsigned short

long, unsigned long, long long…

float

double

double, long double

float _Complex, double _Complex, long double _Complex

float _Imaginary, double _Imaginary, long doublé _Imaginary

*definidos a partir del compilador C99

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES

- C incluye varias clases de **operadores** para formar **expresiones** combinando constantes, variables y funciones

- **Definición general**

  Aritméticos **+, -, *, /, %** (division modulo)

  Relacionales

  < , **>**, **<=** , **>=**, **==** (igual), **!=** (distinto)

  Lógicos

  && corresponde con el AND lógico

  || corresponde con el OR lógico

  ! corresponde con NOT

  NOTA *: el* 0 *representa en C falso. 1 repersenta cierto (o cualquier valor distinto de cero se considera por defecto cierto).*

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES: PRECEDENCIA

- La precedencia de los operadores relacionales es menor que la de los operadores aritméticos.

Por ejemplo, i + j < k - 1 significa (i + j) <(k - 1).

- Los operadores relacionales son asociativos a la izquierda.

La expresión:

i <j <k la acepta el compilador, pero no prueba si j se encuentra entre i y k.

Como el operador < es asociativo a la izquierda, esta expresión es equivalente a

(i <j) <k *(ojo!! i<j, es 1 o 0)*

La expresión correcta es i <j && j <k.

- Los operadores de igualdad tienen una prioridad más baja que los operadores relacionales, por lo tanto la expresión

i <j == j <k es equivalente a (i <j) == (j <k)

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES, PRECEDENCIA. Más ejemplos

i + j * k	es equivalente a	i + (j * k)
-i * -j	es equivalente a	(-i) * (-j)
+i + j / k	es equivalente a	(+i) + (j / k)
i - j – k	es equivalente a	(i - j) - k
i * j / k	es equivalente a	(i * j) / k

- **Los operadores aritméticos unarios (+ y -) son asociativos a la derecha, por lo que:**

  - + i es equivalente a - (+ i)

- **Dado que la asignación es un operador, se pueden encadenar varias asignaciones juntas (asociativo a la derecha):**

  i = j = k = 0 es equivalente a i = (j = (k = 0))

i = 2; j = i * i++; *¿ que valor tiene j después de la ejecución?*

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES LÓGICOS BINARIOS

C incluye operadores que permiten manejar cada bit, de forma individual, en una palabra de memoria.

Los operadores binarios lógicos son:

& (AND binario)
| (OR binario)
^ (XOR)

Supongamos que a=13, el valor binario sería 0…0 **1101**, para b=6 tendríamos 0…0 **0110.** Si hacemos el AND, OR y XOR tendríamos:

	1101		1101		1101
	0110		0110		0110
AND	0100 → 4	OR	1111 → 15	XOR	1011 → 11

Entrada /Salida

Función `printf`

- La función printf tiene asociado un formato: ***"format string",*** seguido de los valores (variables que tienen que salir por pantalla).

printf(*string*, *expr*1, *expr*2, …);

- La cadena de formato puede contener tanto caracteres ordinarios como especificaciones de conversión, que comienzan con el carácter %.
- Una especificación de conversión es un marcador de posición que representa un valor que se debe completar durante la impresión.
- % d se usa para valores int
- % f ser usa para valores flotantes
- % c para caracteres …

Función `printf`

- Los caracteres ordinarios en una cadena de formato se imprimen como aparecen en la cadena.

Ejemplo:

```
int i, j;
float x, y;

i = 10;
j = 20;
x = 12.3456f;
y = 1234.0f;

printf ("i =% d, j =% d, x =% f, y =% f \ n", i, j, x, y);
```

Salida:

i = 10, j = 20, x = 12.345600, y = 1234.000000

Printf. Conversión

- Los compiladores no están obligados a verificar que el número de especificaciones de conversión en una cadena de formato coincida con el número de elementos de salida.

```
printf ("% d% d \ n", i);      /* INCORRECTO */
printf ("% d \ n", i, j);      /* INCORRECTO */
```

- Los compiladores no están obligados a verificar que una especificación de conversión sea apropiada. Si el programador usa una especificación incorrecta, el programa producirá una salida sin sentido:

```
printf ("% f% d \ n", i, x);   /*INCORRECTO */
```

(i definido como int y x como float)

- No se puede romper el ***"format string"*** con un retorno de carro

```
printf (" Los resultados son: %f,  %d\n",
Valor1, Valor2)                /* INCORRECTO */
```

Función `printf`

Una especificación de conversión puede tener la forma % m.pX o % -m.pX, donde m y p son constantes enteras y X es una letra.

Tanto m y p son opcionales; si se omite p, también se elimina el punto que separa m y p.

En la especificación de conversión% 10.2f, m es 10, p es 2 y X es f.

En la especificación% 10f, m es 10 y falta p, pero en la especificación% .2f, p es 2 y falta m.

Enteros

- El ancho de campo mínimo, m, especifica el número mínimo de caracteres para imprimir.
- Si el valor a imprimir requiere menos de m caracteres, se justifica a la derecha dentro del campo.
- % 4d muestra el número 123 como • 123. (• representa el carácter de espacio.)
- Si el valor que se va a imprimir requiere más de m caracteres, el ancho del campo se expande automáticamente al tamaño necesario.
- Poner un signo menos delante de m causa justificación a la izquierda.
- La especificación% -4d mostraría 123 como 123 •.
- p indica el número mínimo de dígitos para mostrar (si es necesario, se agregan ceros adicionales al comienzo del número).
- Si se omite p, se supone que es 1.

Punto flotante

Los especificadores de conversión para números de punto flotante son:

- e - Formato exponencial. p indica cuántos dígitos deben aparecer después del punto decimal (el valor predeterminado es 6). Si p es 0, no se muestra ningún punto decimal.
- f - Formato "decimal fijo". p tiene el mismo significado que para el especificador e.
- g: formato exponencial o formato decimal fijo, según el tamaño del número. p indica el número máximo de dígitos significativos que se mostrarán. La conversión g no mostrará ceros finales. Si el número no tiene dígitos después del punto decimal, g no muestra el punto decimal.

Programa ejemplo printf

```
#include <stdio.h>

    int main(void)
     {
        int i;
        float x;

        i = 40;
        x = 337.21f;
       printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
       printf("|%10.3f|%10.3e|%-10g|\n", x, x, x);
      return 0;
    }
```

Salida:

```
|40|   40|40   |  040|
|  337.210|3.372e+02|337.21    |
```

Secuencias de “escape”

El código **\n** que se usa en las cadenas de formato se llama secuencia de escape.
Las secuencias de escape permiten que las cadenas contengan caracteres no imprimibles (control) y caracteres que tienen un significado especial.
Algunas secuencias de escape son:

Alert (bell)	**\a**
Backspace	**\b**
New line	**\n**
Horizontal tab	**\t**

Se pueden usar varias secuencias como:
printf("Item\t Unidad\tCompra\n\tPrecio\tFecha\n");

Al ejecutar esta instrucción se imprime un encabezado de dos líneas:

Item Unidad Compra
Precio Fecha

Secuencias de “escape”

Otra secuencia de escape común es \", que representa el carácter":

```
printf ("\" Hello! \ "");
/ * imprime "¡Hola!" * /
```

Para imprimir un solo \ carácter, se ponen dos \ caracteres en la cadena:

```
printf ("\\");
/ * imprime un \ carácter * /
```

Función. Scanf

scanf lee la entrada de acuerdo a un formato particular.

Una cadena de formato scanf puede contener tanto caracteres ordinarios como especificaciones de conversión. Las conversiones permitidas con scanf son esencialmente las mismas que las utilizadas por **printf( )**

Ejemplo:

int i, j;

float x, y;

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

Ejemplo de entrada:

1 -20 .3 -4.0e3

scanf asignará 1, –20, 0.3 y –4000.0 a i, j, x e y, respectivamente.

Función. Scanf

scanf intenta hacer coincidir grupos de caracteres de entrada con las especificaciones de conversión en la cadena de formato.

Para cada especificación de conversión, scanf intenta localizar un elemento del tipo apropiado en los datos de entrada, omitiendo espacios en blanco si es necesario, scanf luego lee el elemento y se detiene cuando alcanza un carácter que no puede pertenecer al elemento.

Si el elemento se leyó correctamente, scanf continúa procesando el resto de la cadena de formato.

Cuando busca un número, scanf ignora los caracteres de espacio en blanco (espacio, tabulación horizontal y vertical, avance de página y nueva línea).

Función. Scanf

Una llamada de scanf que lee cuatro números sería:

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

Los números pueden estar en una línea o repartidos en varias líneas:

1 9

-20 .3

-4.0e3

Ejemplo de entrada:

1-20.3-4.0e3¤

La llamada de scanf es la misma que antes:

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

Aquí es cómo scanf procesaría la nueva entrada:

%d. Almacena 1 en i y devuelve el carácter.

%d. Almacena –20 en j y pone el caracter de vuelta.

%f. Almacena 0.3 en x y devuelve el carácter.

%f. Almacena –4.0 × 103 en y y devuelve el carácter de nueva línea.

Función. Scanf

Si la cadena de formato es "% d/% d" y la entrada es • 5 / • 96, scanf se realiza correctamente. (• *representa blanco*)

Si la entrada es • 5 • / • 96, scanf falla, porque / en la cadena de formato no coincide con el espacio en la entrada.

Para permitir espacios después del primer número, el formato correcto es "% d /% d" .

Función. Scanf

Mas ejemplos que combinan especificaciones de conversión, caracteres de espacio en blanco y caracteres que no son espacio en blanco (ojo! *experimentad*!):

	Entrada	Resultado
n = scanf ("%d%d", & i, & j);	12 •, • 34¤	n=1 (leidos) i: 12 j: sin cambios
n = scanf ("%d,%d", & i, & j);	12 •, • 34¤	n: 1 i: 12 j: sin cambios
n = scanf ("%d• ,%d", & i, & j);	12 •, • 34¤	n: 2 i: 12 j: 34
n = scanf ("%d, •% d", & i, & j);	12 •, • 34¤	n: 1 i: 12 j: sin cambios

Tipos de Datos /Operadores II

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

RECORDEMOS

C incluye operadores que permiten manejar cada bit, de forma individual, en una palabra de memoria.

Los operadores binarios lógicos son:

& (AND binario)
| (OR binario)
^ (XOR)

Supongamos que a=13, el valor binario sería 0…0 **1101**, para b=6 tendríamos 0…0 **0110.** Si hacemos el AND, OR y XOR tendríamos:

	1101		1101		1101
	0110		0110		0110
AND	0100 → 4	OR	1111 → 15	XOR	1011 → 11

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

OPERADORES LÓGICOS BINARIOS

```
#include <stdio.h>
#include <stdlib.h>

int main()
  {
    int a, b, OPand, OPor, OPxor;

    a=13;
    b=6;
    OPand=a&b;    //  operación AND
    OPor=a|b;     //   operación OR
    OPxor=a^b;    //   operación XOR
    printf("el AND lógico es: %d, el OR es:  %d y el XOR es: %d", OPand, OPor, OPxor);
   return 0;
}
```

La salida es:

```
el AND lógico es: 4, el OR es: 15 y el XOR es: 11
```

CONCEPTOS BÁSICOS

TIPOS DE DATOS Y OPERADORES

EFECTOS SECUNDARIOS. Side Effects

- Se dice que un operador que modifica uno de sus operandos tiene un "side effects"

```
int i;
float f;
f = i = 33.3f;
```

En este ejemplo tenemos que a **i** se le asigna el valor 33 y después a **f** se le asigna 33.0 (no 33.3).

```
i = 1;
k = 1+( j = i); /* es válida la asignación pero.. */
printf("%d %d %d\n", i, j, k);
```

CONCEPTOS BÁSICOS

CONVERSION DE TIPOS

Cuando tenemos una expressión de asignación o una operación, podemos tener operandos de distinto tipo. En estas situaciones se realiza una conversion de tipos , que puede ser:

implicita, si la realiza el propio compilador

explicita, si la realiza en programador (casting)

Nos podemos encontrar por ejemplo, con:

```
char n;
int a, b, c;
float r, s, t;
        a = 10;
        b = 100;
        r = 1000;       /* ¿Qué tipos corresponden? */
        s = r + a;
        c = r + b;
        c = n + a + r;
```

CONCEPTOS BÁSICOS

CONVERSION DE TIPOS: implícita

- La conversion implícita se rige por las reglas de la version del compilador, en general tenemos:

Si algún operando es de tipo long double, el otro se convertirá a long double.

Si algún operando es de tipo double, el otro se convertirá a double.

Si algún operando es de tipo float, el otro se convertirá a float.

Si algún operando es de tipo unsigned long long, el otro se convertirá a unsigned long long.

Si algún operando es de tipo long long, el otro se convertirá a long long.

Si algún operando es de tipo unsigned long, el otro se convertirá a unsigned long.

Si algún operando es de tipo long, el otro se convertirá a long.

Si algún operando es de tipo unsigned int, el otro se convertirá a unsigned int.

Del mismo modo, cuando la variable receptora (asignación) es de distinto tipo que el resultado de la expresión de la derecha se aplica la misma conversión de tipos.

CONCEPTOS BÁSICOS

CONVERSION DE TIPOS: explícita, casting

- Para realizar una conversión explícita, "casting" (cast) es tan sencillo como poner **(tipo de dato)** delante de la expresión o variable a convertir

Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main( )
   {
   int a=10;
   printf("%d, %f, %e \n",  a, (float)a, (double)a );
   return 0;
  }
```

La salida sería:

10, 10.000000, 1.000000e+001

CONCEPTOS BÁSICOS

CONVERSION DE TIPOS: explícita, casting

Ejemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
    {
        int a=10, b=3;
        float c;
        c=a/b;
        printf("%f\n", (float )c);
        c=(float)a/b; /* es equivalente a poner a/(float)b */
        printf("%f\n", c);
        return 0;
    }
```

La salida es: 3,000000

3,333333

CONCEPTOS BÁSICOS

DEFINICIÓN DE TIPOS. Typedef

Para crear un nuevo tipo de datos en C, se utiliza la palabra reservada **typedef**

typedef *tipo_de_datos nuevo_tipo;*

typedef int Bool;

Bool pasa a ser así un nuevo tipo de dato entero y que podemos utilizar de forma similar en definiciones, asignaciones, etc que float, char, int, double…

Ejemplo:

Bool encendido;

NOTA: veremos que typedef tiene gran utilidad en la creación de tipos de datos estructurados

CONCEPTOS BÁSICOS

FUNCION SIZEOF( )

El operador **SIZEOF( )**, devuelve el tamaño en bytes que ocupa un determinado tipo de datos o una variable.

```
#include <stdio.h>
int main()
{
   int a;
   printf(" Un int ocupa %d bytes\n", sizeof(a)); /* usando una variable */
   printf(" Un char  ocupa %d bytes\n", sizeof(char));
   printf(" Un float ocupa %d bytes\n", sizeof(float));
   printf(" Un double ocupa %d bytes\n", sizeof(double));
   printf(" Un short ocupa %d bytes\n", sizeof(short));
   printf(" Un long ocupa %d bytes\n", sizeof(long));
   printf(" Un long double ocupa %d bytes\n", sizeof(long double));
   …….
  return 0;
}
```

CONCEPTOS BÁSICOS

Rangos según tipo de datos

- **Rangos en una máquina de 64-bits:**

Tipo	*Valor mas pequeño*	*Máximo Valor*
`short int`	–32,768	32,767
`unsigned short int`	0	65,535
`int`	–2,147,483,648	2,147,483,647
`unsigned int`	0	4,294,967,295
`long int`	-2^{63}	$2^{63}-1$
`unsigned long int`	0	$2^{64}-1$

En **<limits.h>** están definidos estos valores

CONCEPTOS BÁSICOS

Punto flotante

- Las características de `float y double` corresponden a un estándar IEEE :

Tipo	*Valor mas pequeño*	*Valor mas grande*	*Precisión*
`float`	`1.17549 × 10-38`	`3.40282 × 1038`	`6 digits`
`double`	`2.22507 × 10-308`	`1.79769 × 10308`	`15 digits`

Pero: puede ocurrir que un compilador concreto de C no respete el IEEE Standard 754 (también conocido como IEC 60559).

Por defecto, las constantes de números en punto flotantes se almacenan como números de doble precisión.

Para indicar que solo se desea una precisión simple se pone a letra F (o f) al final de la constante (por ejemplo, 57.0f). Para indicar que una constante debe almacenarse en un formato doble largo, se pone la letra L (o l) al final.

CONCEPTOS BÁSICOS

TIPO CHAR.

Cuando aparece un carácter (tipo char) en un cálculo, C usa su valor entero.

Consideremos este programa

```
#include <stdio.h>
        int main()
         {
            int i;
            char ch;
            i='a';          /* i definida como un entero almacena el valor ascii de 'a', 97 */
            ch= 65;       /* ch definida como un char asocia el valor acii 65, esto es, 'A' */
            printf(" %c  %c  %d  %d   %c\n" , i, ch, i, ch);
            return 0;
        }
```

La salida de este programa es: 'a' 'A' 97 65 . El compilador C nos permite trabajar a la vez como int y como char un carácter

si pedimos imprimir ch+1 como char (%c) nos saldría 'B'

Y podemos escribir también, por lo tanto:

```
for (ch = 'A'; ch <= 'Z'; ch++){
…..
   }
```

Notaciones comprimidas

Notaciones comprimidas

En programación son comunes las asignaciones que utilizan el valor antiguo de una variable para calcular su nuevo valor .

Ejemplo:

i = i + 2;

Usando el operador de asignación “compuesto”, simplemente escribimos:

i + = 2; / * por lo tanto sería igual que i = i + 2; * /

- Para el resto de operadores:
  v + = e agrega v a e, almacenando el resultado en v
  v - = e resta e de v, almacenando el resultado en v
  v * = e multiplica v por e, almacenando el resultado en v
  v / = e divide v por e, almacenando el resultado en v
  v% = e calcula el resto cuando v se divide por e, almacenando el resultado en v

Notaciones comprimidas

Dos de las operaciones más comunes en una variable son "incrementar" (sumar 1) y "disminuir" (restar 1):

```
i = i + 1;
j = j - 1;
```

Incrementar y disminuir se puede hacer usando los operadores de asignación compuesta:

```
i + = 1;
J - = 1;
```

C proporciona además dos operadores especiales **++** (incremento) y **--** (decremento).

El operador **++** agrega 1 (i++) a su operando. El **--** operador resta 1 (i--).

Tienen efectos secundarios: modifican los valores de sus operandos.

Notaciones comprimidas

La expresión **++ i** significa "incrementar i inmediatamente", mientras que **i++** significa "usar el valor antiguo de i por ahora, pero incrementar i más tarde". El operador **- -** funciona de igual manera:

```
i = 1;
printf("i es %d\n", --i);   /* imprime "i es 0" */
printf("i es %d\n", i);     /* imprime "i es 0" */
i = 1;
printf("i es %d\n", i--);   /* imprime "i es 1" */
printf("i es %d\n", i);     /* imprime "i es 0" */
```

Cuando se usa ++ o - - más de una vez en la misma expresión, el resultado puede ser difícil de entender.

```
i = 1;
j = 2;
k = ++ i + j ++;
```

Los valores finales de i, j y k son 2, 3 y 4, respectivamente.

Conceptos básicos de macros

Conceptos básicos de macros /directivas

- La estructura básica de un programa en C es:

```
directivas
int main()
  {
     statements
  }
```

- Antes de compilar un programa en C, primero lo edita un preprocesador.
- Los comandos destinados al preprocesador se denominan directivas.

  Ejemplo:

  #include <stdio.h>

  <stdio.h> es un encabezado que contiene información sobre la biblioteca de E/S estándar de C

- Las directivas siempre comienzan con un # carácter, no hay punto y coma ni otro marcador especial al final.

Conceptos básicos de macros /directivas

- ***macro definition .*** Mediante la directiva **#define**, podemos definer costantes que vamos a utilizar en el resto del programa. Por ejemplo, si empleamos varias veces el valor de **π**, podemos escribir:

```
#include<stdio.h>
#define PI 3.1415926
main( )
  {
     float radio=0, circulo=0;
     scanf("Introduzca el radio del círculo %f: ", &radio);
     circulo=2*PI*radio;
     printf("La longitude del círculo es de: %f\n", circulo);
     return 0
  }
```

El compilador en la primera "pasada" (preprocesador) sustituye PI por su valor en todo el texto.

Conceptos básicos de macros /directivas

NOTA

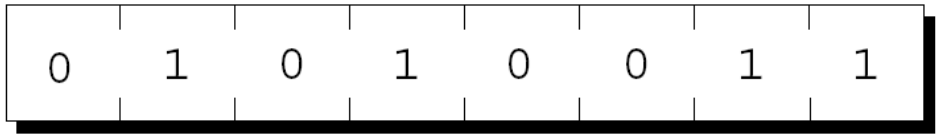
Se pueden construir “defines” más complejas como:

```
#include <stdio.h>
#define PI 3.1415926
#define area(x) PI*x*x
int main()
{
    float radio=0.0;
    printf("Escriba el radio de la circunferencia", &radio);
    scanf("%f", &radio);
    printf("La superficie para un radio de:=%f, es: %f\n", radio, area(radio));
    return 0;
}
```

Introducción a los Punteros

Variables tipo Puntero

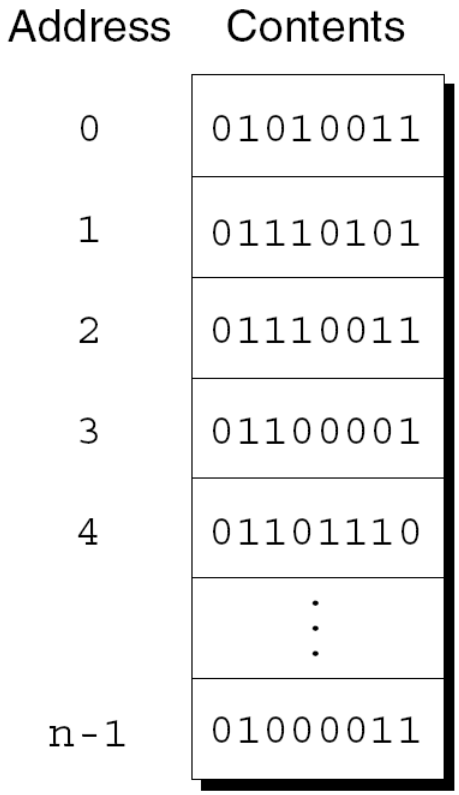
- El primer paso para comprender los punteros es visualizar lo que representan a nivel de máquina.
- En la mayoría de las computadoras modernas, la memoria principal se divide en bytes, con cada byte capaz de almacenar ocho bits de información



- Cada byte tiene una única dirección.

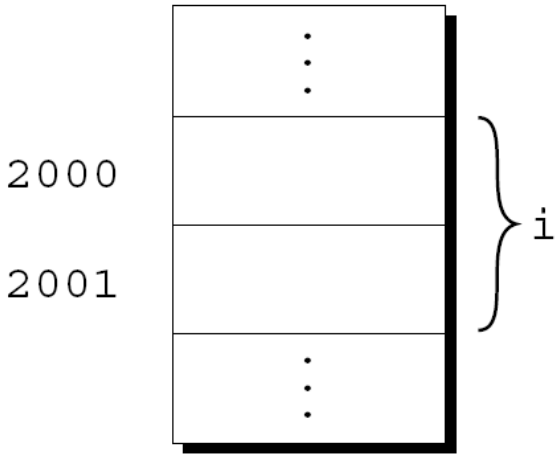
Variables tipo Puntero

- Si hay n bytes en la memoria, podemos pensar en las direcciones como números que van de 0 a n - 1:



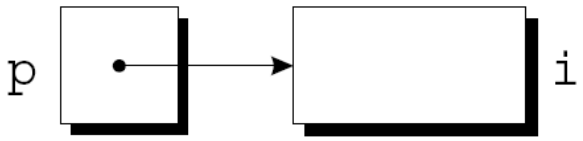
Variables Puntero

- Cada variable en un programa ocupa uno o más bytes de memoria.
- La dirección del primer byte se dice que es la dirección de la variable.
- En la siguiente figura, la dirección de la variable i es 2000:



Variables Puntero

- Las direcciones se pueden almacenar en variables de puntero especiales.
- Cuando almacenamos la dirección de una variable i en la variable de puntero p, decimos que p "apunta a" i.
  Una representación gráfica:



Declaración de Punteros

- Cuando se declara una variable de puntero, su nombre debe ir precedido por un asterisco:

```
int *p;
```

- p es una variable de puntero capaz de apuntar a objetos de tipo int.
- Se usa el término objeto en lugar de variable, ya que p podría apuntar a un área de la memoria que no pertenece a una variable.

Declaración Punteros (Pointer)

- Las variables de puntero pueden aparecer en declaraciones junto con otras variables:

```
int i, j, a[10], b[20], *p, *q;
```

- C requiere que cada variable de puntero apunte solo a objetos de un tipo particular (el tipo referenciado):

```
int *p;      /* solo apunta a integer   */
double *q;   /* double*/
char *r;     /* characteres */
```

- No hay restricciones sobre lo cual puede ser el tipo referenciado.

Los Operadores de Dirección e Indirección

C proporciona un par de operadores diseñados específicamente para su uso con punteros.

- Para encontrar la dirección de una variable, usamos el operador **&** (dirección).

- Para obtener acceso al objeto al que apunta un puntero (contenido), usamos el operador ***** (indirección).

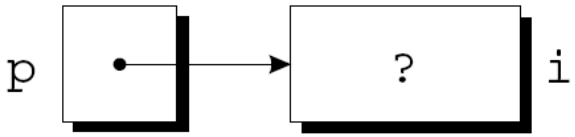
Los Operadores de Dirección e Indirección

- Una forma de inicializar una variable de puntero es asignarle la dirección de una variable:

```
int i, *p;
…
p = &i;
```

- Asignar la dirección de i a la variable p hace que p apunte a i:
- También es posible inicializar una variable de puntero en el momento en que se declara:

int i, *p = &i;



Los Operadores de Dirección e Indirección

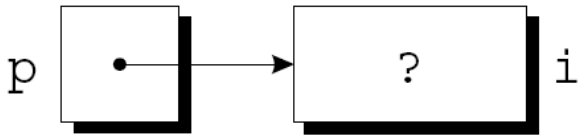
- Una vez que una variable de puntero apunta a un objeto, podemos usar el operador * (indirección) para acceder a lo que está almacenado en el objeto.

- Si p apunta a i, podemos imprimir el valor de i de la siguiente manera:

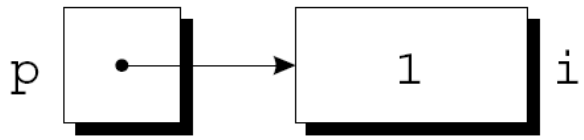
```
printf("%d\n", *p);
```

Los Operadores de Dirección e Indirección

```
p = &i;
```

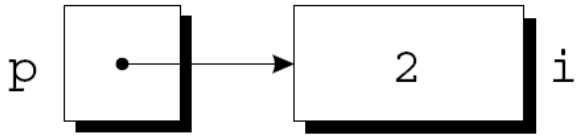



```
i = 1;
```




```
printf("%d\n", i);     /* imprime 1 */
printf("%d\n", *p);    /* imprime 1 */

*p = 2;
```



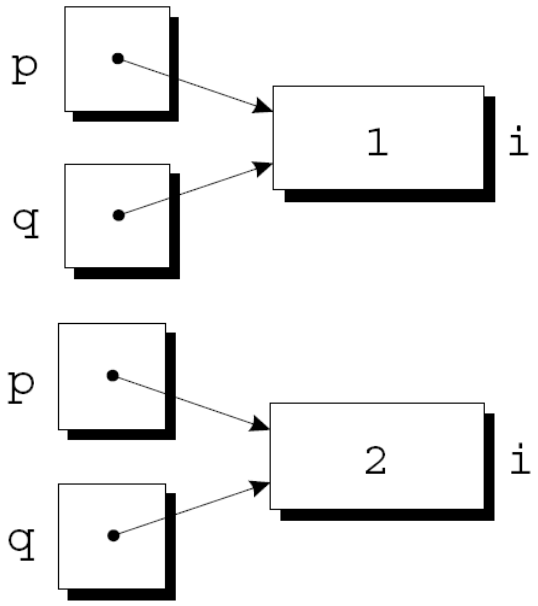

```
printf("%d\n", i);     /* imprime 2 */
printf("%d\n", *p);    /* imprime 2 */
```

Los Operadores de Dirección e Indirección

- Si `p` y `q` apuntan a `i`, podemos cambiar i asignando un nuevo valor a: *p o *q:

```
*p = 1;
```

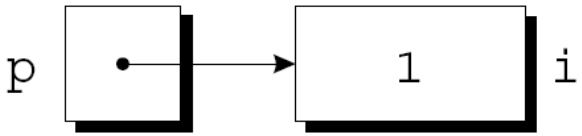
```
*q = 2;
```

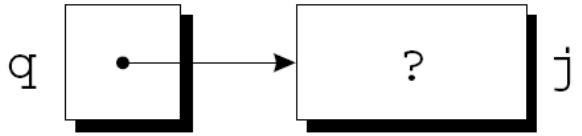


- Cualquier número de variables de puntero puede apuntar al mismo objeto.

Asignación entre Punteros

```
p = &i;
q = &j;
i = 1;
```






```
*q = *p;
```