)*+,*- . -/$%&’0#1*2/12*-3-’"&’45!6’7

!"#$%&’(

8-9-":’ ; "&<&3"=’3"’>:-&+’8+=-#

!"#$%&$ ’ "(&) !" *(+)% ’ ,&*-$. "/"-&%0(*-$

"12345623786941345:;<98=64<6>?498=3@=86A

-73628B34-?55?@14C.D(-DE$4FGH

Competencias y Habilidades

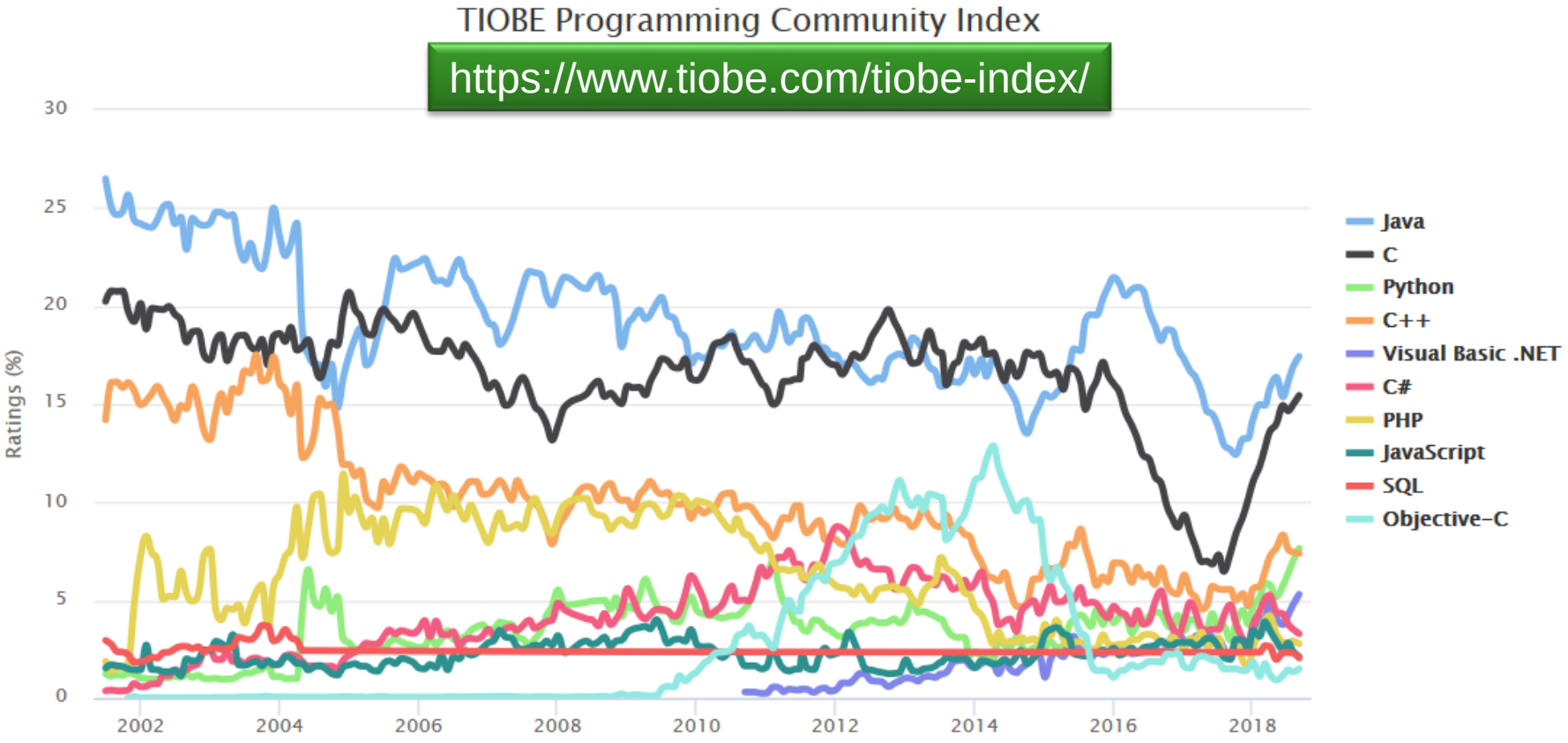
Introducción al Lenguaje ANSI C

Bibliografía:

- Kernighan, B., Ritchie, D. El lenguaje de programación (ANSI) C. 2ª ed. PrenticeHall-Pearson.
- Deitel, P. & H. C for programmers. Prentice Hall. 2013.

Índice

1. Motivación: Mercado



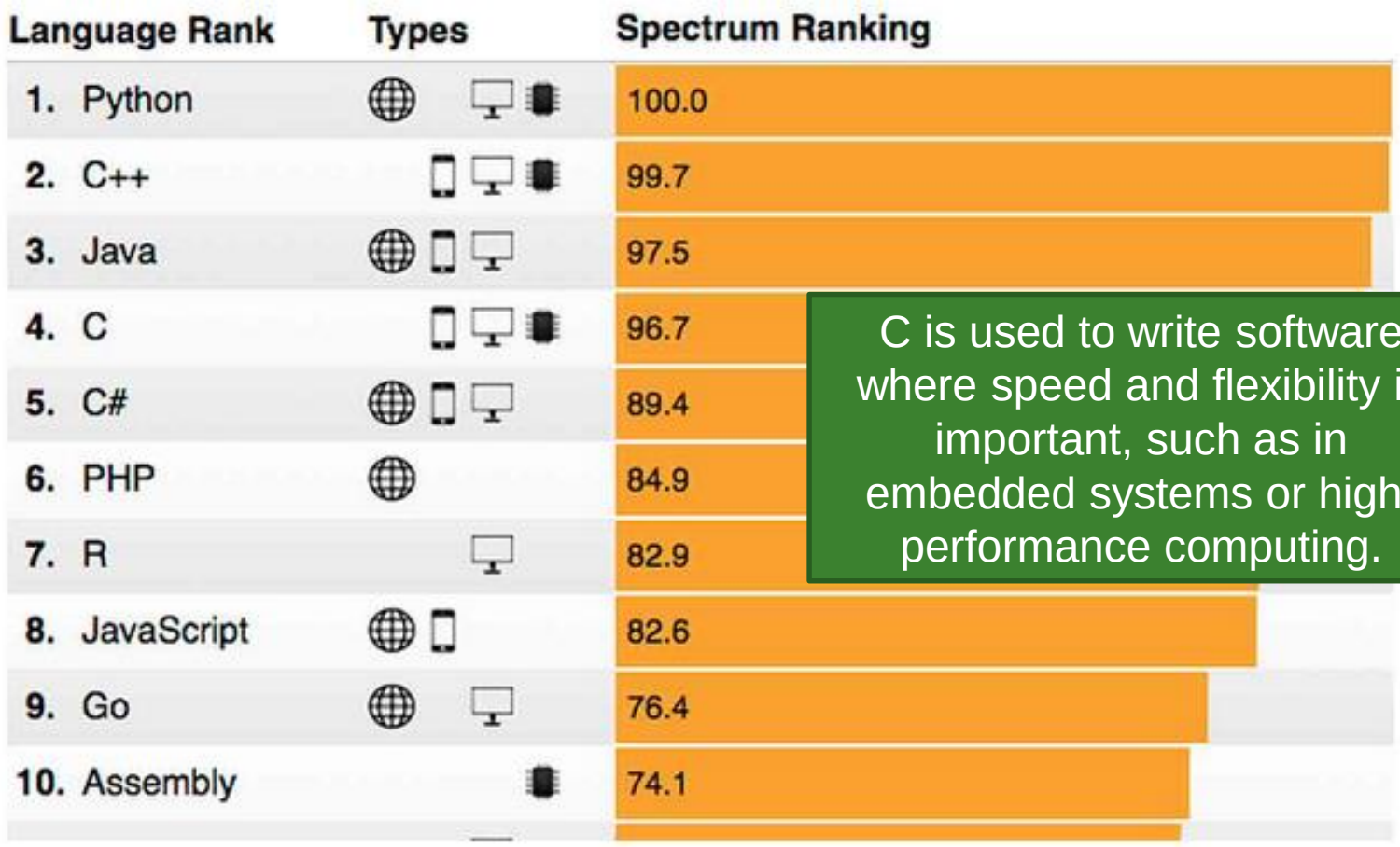
Java 17% / C 15% + C++ 8% + c# 3% + obj. C 1% = 27%

1. Motivación: Mercado

Sep 2018	Sep 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.436%	+4.75%
2	2		C	15.447%	+8.06%
3	5	▲	Python	7.653%	+4.67%
4	3	▼	C++	7.394%	+1.83%
5	8	▲	Visual Basic .NET	5.308%	+3.33%
6	4	▼	C#	3.295%	-1.48%
7	6	▼	PHP	2.775%	+0.57%
8	7	▼	JavaScript	2.131%	+0.11%
9	-	⇈	SQL	2.062%	+2.06%
10	18	⇈	Objective-C	1.509%	+0.00%
11	12	▲	Delphi/Object Pascal	1.292%	-0.49%
12	10	▼	Ruby	1.291%	-0.64%
13	16	▲	MATLAB	1.276%	-0.35%
14	15	▲	Assembly language	1.232%	-0.41%

https://www.tiobe.com/tiobe-index/

1. Motivación: Mercado



C is used to write software where speed and flexibility is important, such as in embedded systems or high-performance computing.

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

1. Introducción: Características

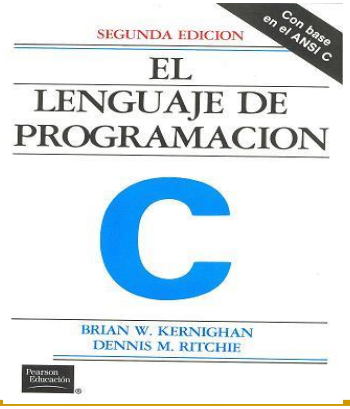
- Lenguaje que se creo para escribir el código de UNIX casi completamente.
- Lenguaje de nivel medio:
  - Cercano al ensamblador.
- Lenguaje estructurado.
- No fuertemente tipado.
- Los compiladores son sencillos y producen código muy eficiente (pero no seguro).
- Se permite la compilación separada.
- Puede resultar poco claro con poca disciplina.

1. Introducción: Versiones

- El desarrollo inicial de C se llevó a cabo por Ken Thompson y Dennis M. Ritchie en los Laboratorios Bell de AT&T entre 1969 y 1973.
  - Se le dio el nombre "C" porque muchas de sus características fueron tomadas de un lenguaje anterior llamado "B".
- En 1978, Ritchie y Brian Kernighan publicaron la primera edición de "*El lenguaje de programación C"*. Este libro fue durante años la especificación del lenguaje.
  - **El C de Kernighan y Ritchie** o simplemente "K&R C" .
  - La segunda edición del libro cubre el estándar ANSI C.
- Debido a los numerosos añadidos y que cada compilador metía nuevas librerías, se trató de estandarizar en 1983.
- En 1989 se ratificó como el "Lenguaje de Programación C" ANSI X3.159-1989. Esta versión del lenguaje se conoce a menudo como **ANSI C**, o a veces como C89.
  - En 1990, el estándar ANSI fue adoptado por ISO: C90.
- La siguiente revisión del estándar fue la ISO 9899:1999 en 1999. Este estándar se denomina habitualmente "**C99**". Se adoptó como estándar ANSI en marzo de 2000. Fue mucho peor acogida.
- La última es el "**C11**" que soporta multithreading.

Introducción: C89

- Cambio de paso de parámetros a función.
- Prototipos de función.
- Funciones void y el tipo de datos void *.
- Funciones que retornaban tipos de datos struct o union (en lugar de punteros).
- Asignación de tipos de datos struct.
- Calificador const, que hace que un objeto sea de sólo lectura.
- Una librería estándar.
- Tipo de dato enumeración: enum.
- ...



Introducción: C99

- Funciones inline.
- Las variables pueden declararse en cualquier sitio (como en C++).
- Muchos tipos de datos, incluyendo long long int (transición de 32 bits a 64 bits), un tipo de datos booleano, y un tipo complex que representa números complejos.
- Arrays (vectores, arreglos) de longitud variable.
- Soporte para comentarios de una línea que empiecen con // (como C++).
- Muchas funciones y headers nuevos.
- ...
- La historia continua, hay un **C11** que da soporte al multithreading.

Índice

1. Elementos lexicográficos y estructura

- El lenguaje está compuesto por “tokens” que son identificadores, operadores y palabras reservadas:
  - Un identificador será un nombre compuesto por letras, dígitos y el carácter "_", que deberá comenzar por una letra (31 c.).
  - Se distingue entre mayúsculas y minúsculas.
  - Las palabras reservadas son 32 (ANSI):

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

1. Elementos lexicográficos y estructura

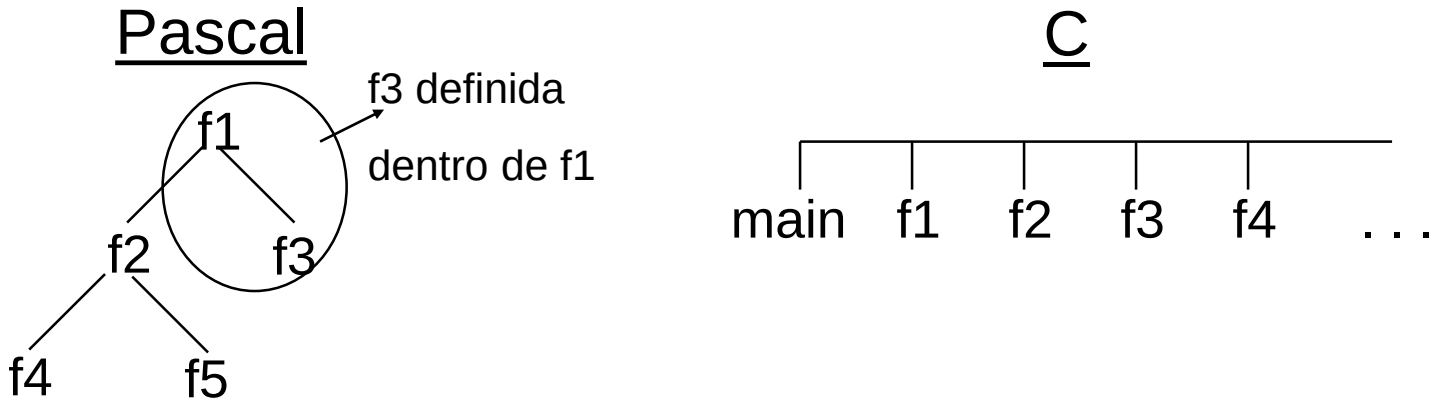
- También existen **delimitadores** que son:
  - Espacio, tab, el ↵ y los comentarios (entre /* */ o de línea //).
- Con tokens y delimitadores se construyen **sentencias** (instrucciones) que acaban con "**;**".
- Además existirán las **declaraciones** de datos.
- Estos dos elementos conforman un bloque (Java y C):

```
{
   <declaraciones>
   <instrucciones>
}
```

- Un programa en C estará compuesto de al menos una **función** -la función main()- compuesta de un bloque.

1. Elementos lexicográficos y estructura

- El C todas las funciones de un programa están al mismo nivel, no se pueden anidar a diferencia de otros lenguajes.



- Hay instrucciones que no son del lenguaje, sino del **preprocesador** y empiezan con "#", típicamente:
  - #define NOMBRE constante
  - #include <cabecera.h> o "fichero.c"

1. Estructura: bloques

Java	C

`import` modulo1.*;
`public class` Clase {
`public static void`
main
(String[ ] args)
{
<declaraciones>
<instrucciones>
}
}
	#include <modulo1.h>
	`int` main()
	{
	<declaraciones>
	<instrucciones>
	}

1. Estructura: comentarios

- En C existen dos tipos de comentarios:
  - Los de bloque (prólogo). /* */
    - Se utilizan para comentar los bloques de código dando un pseudocódigo o más abstracto.
    - Exclusión de código.
  - Los de fin de línea (inline). //
    - Explicar alguna línea de código compleja.
    - Fin de función.

- Existe controversia y muchas escuelas:
  1. Si se necesitan comentarios el código está mal escrito (!!!). Sólo tienen que ser abstracciones.
  2. Describir lo que hace un bloque de código, entradas y salidas.
  3. Describir el algoritmo que se utiliza y porqué.
  4. Recursos. Arte ASCII (Logotipos y diagramas de flujo), derechos de autor, fecha de creación, versión del producto, contacto con el programador …)

1. Estructura: ejemplo más sencillo

```
#include <stdio.h>
int main()
{
   printf("hola\n"); // printf escribe en pantalla
   return (0);
}
-----------------------------------------
public class Hola{
   public static void main(String[] args)
   {
      System.out.println("hola");
      // println escribe
   }
}
```

Ejercicio 1: Aspecto de un programa C

```
#include <stdio.h>                    /* fichero de cabecera para el preprocesador */
#define CONVERSION 166.386                    /* constante del preprocesador */


int euros = 4;    /* variable global entera inicializada */

main ()       /* Pasa 4 euros a pesetas */
{
              int pesetas;    /* variable local a main */


              pesetas = CONVERSION * euros;       /* conversiones de tipo */
              printf("Hay %d pesetas en %d euros \n", pesetas, euros);     /*salida*/
              pinta();                            /* llamada a función externa */
}


pinta()   /* cabecera de función */
{
              printf("Solo hago esto \n");
}
```

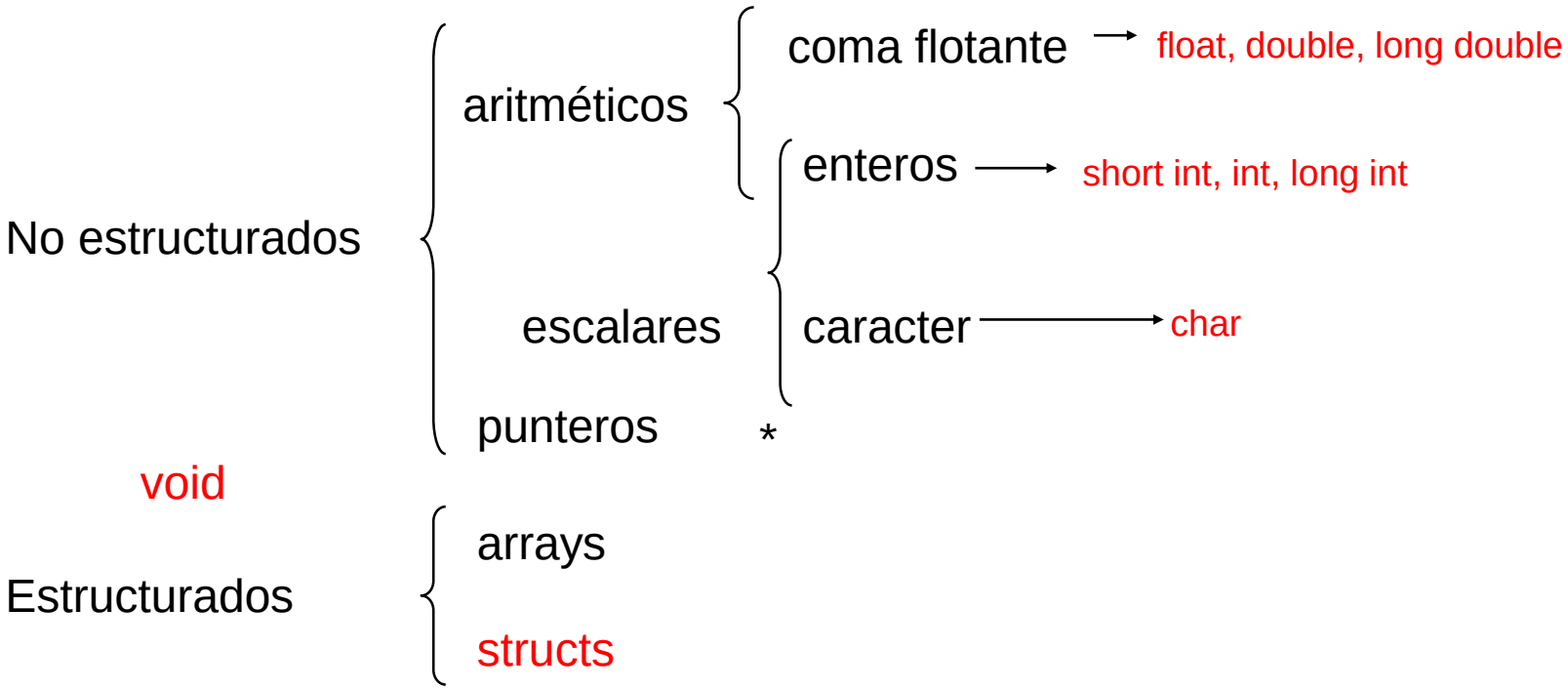
Ejercicio 1: Aspecto de un programa C

1. Bájate el programa 1.c y guárdalo en tu directorio.
2. Ábrelo con un editor y examínalo e identifica sus partes. Mira el estilo de identación.
3. Compílalo con el comando `$ gcc 1.c`.
4. Comprueba que tienes un ejecutable y su nombre.
5. Ahora prueba con `$ gcc -ansi 1.c`
6. Ahora prueba con `$ gcc -Wall 1.c` y observa los warnings (no son errores).
7. Usa el 2.c para ver que desaparecen.
8. ¿Cuáles son las diferencias?
9. Sólo por curiosidad mira las opciones de gcc (man gcc).

Índice

1. Tipos de datos

- Existen dos clases de datos: las constantes y las variables.
- Y de ellos varios tipos de datos:



1. Tipos de datos: extras del ANSI

- Y los enteros se pueden usar con o sin signo.
  - `unsigned long int  o  (signed)`
- Además en el ANSI C, además de los tipos clásicos, existen:
  - Los tipos de datos enumerados:
    - `enum colores {verde, rojo, amarillo, violeta, azul, naranja};`
    - `enum niveles {bajo = 100, medio = 500, alto = 2000};`
    - `enum logico {falso=0, verdadero=1};`
- Siempre se pueden "definir" nuevos tipos con **typedef**:

```
typedef   float     real;
typedef   char      *string;
typedef   struct complejo {float real;   float imag;};
```

- En C99 existen otros tipos de datos como extensiones como lógicos (bool) o complejos (complex).

1. Tipos de datos: Comparación

Java	C	Otros tipos C
byte short int long	signed char short, short int int long, long int long long	char unsigned short int unsigned int unsigned long int unsigned long long
boolean	(se usa int) o bool en C99	
char	char	
float double	float double long double	float _Complex c99 double _Complex long double _Complex
String	char[], char*	

1. Tipos de datos: constantes

- Por cada tipo de dato existirá una forma de poner sus constantes para hacer inicializaciones y asignaciones:
  - Enteras: 23
    - Larga: 22L
    - Unsigned: 100u
    - Octal: 023
    - Hexadecimal: 0X43
  - Flotantes: 3.1416 100.
    - Flotante: .2F .2f
    - Científica: 4e16 ó .8E-5
    - Double: -4.3e-5L 5.4e6l
- Si queremos saber el tamaño de un dato tenemos el operador `sizeof ( )` .

1. Tipos de datos: constantes

- Cuando nos referimos a datos de tipo carácter usaremos apóstrofes ‘ ’ y comillas “ ” en cadenas de caracteres (strings).
- Hay caracteres especiales (definidos) a los que nos referimos con ‘\letra’:

1. Carácter: '3' ‘t’
2. Código ASCII: '\007‘ '\x7‘
3. Secuencia de escape (de caracteres ASCII de control):

\n nueva línea	\f salto de página	\0 nulo
\t tabulador	\\ barra atrás	\v tab vertical
\b retroceso	\' apóstrofe	\a alerta
\r retorno de carro	\" comillas	

1. Declaraciones: Comparación

- C

```
int lower;
char c, resp='a'; // dos variables de tipo char
int i=0;          // variable inicializada
const float eps=1.0e-5; // constante
#define MAX 8           // otra constante
```

- Java

```
int lower;
char c,resp;          // dos variables de tipo char
int i=0;              // variable inicializada
final float eps=1.0e-5; // constante
final int max=8;         // otra constante
```

Ejercicio 2: Declaraciones de datos

1. Bájate el programa 3.c y guárdalo en tu directorio.
2. Ábrelo con un editor y examínalo y comprende los tipos de inicializaciones que hay.
3. ¿Cuánto vale b?
4. Cómo lo pondrías.

1. Expresiones: Operandos y operadores

- Una expresión estará compuesta de **operandos** y **operadores**.
- Los operandos tendrán un tipo de dato definido.
- Los operadores actuarán sobre los operandos y tendrán su tipo.
- Para evaluar una expresión se tiene que definir el orden de hacerlo:
  - Asociatividad: A qué número de operandos afectan y su evaluación (izquierda a derecha).
    - Primarios: Los que “encierran”.
    - Unarios: A un operando.
    - Binarios: A dos operandos.
  - Precedencia:
    - Define el “orden” de ejecución.

1. Operadores aritméticos

Operador	Símbolo	Asociatividad	Precedencia
Adición	+	Binario	4º
Substracción	-	Binario	4º
Incremento	++	Unario	2º prefijo
Decremento	--	Unario	2º prefijo
Signo	-	Unario	2º
Multiplicación	*	Binario	3º
División	/	Binario	3º
Resto	%	Binario	3º
Paréntesis	( )	Primario	1º
Asignación	= += -= *= /= %=	Binario	5º

- La precedencia es relativa.
- Ejemplo (en ANSI C de izquierda a derecha).

```
valor = (operando1 + operando2) * operando3;
respuesta = num / 2 + 3 * (1 + num++);      //indefinido  en C K&R
respuesta = num / 2 + 3 * (++num + 1);      //indefinido  en C K&R
```

Ejercicio 3: Expresiones

1. Bájate el programa 4.c y guárdalo en tu directorio.
2. Ábrelo con un editor y examínalo y comprende las operaciones que hay.
3. Escribe cuanto valen las variables izda y dcha en cada paso.
4. Ejecuta el programa y mira si has acertado.
5. Prueba que en ANSI C la expresión anterior no estaba indefinida.

2. Conversión de tipos

- Los operadores aritméticos están sobrecargados.
- Implícita (la que permite el lenguaje –MAL–):
  - flotante = 3.0 + 1/2; // cuanto vale ?
  - flotante = 3.0 + 1.0/2;
  - Promoción o pérdida de rango:
    - long double -> double -> float -> long -> int -> short -> char
- Por asignación:
  - entero = 3.9; //valdrá 3
- Explícita. Existe otro operador para realizar la conversión (type cast), con el uso de paréntesis encerrando el tipo de dato destino:
  - flotante = 3.0 + (float)1/2;

Ejercicio 4: Expresiones

1. Bájate el programa 5.c y guárdalo en tu directorio.
2. Ábrelo con un editor y examínalo y comprende las operaciones que hay.
3. Escribe cuanto valen las variables f , izda y c en cada paso.
4. Ejecuta el programa y mira si has acertado.

1. Entrada/Salida estándar: stdio.h

- Es necesario incluir la cabecera stdio.h `#include <stdio.h>`
- Dos funciones básicas de salida y entrada:
  - printf (control, lista de variables);
  - scanf (control, lista de **direcciones** de variables);
- Control es una tira de caracteres (entre comillas) que nos indica que queremos en la entrada/salida.
- En la tira de control se incluirán los **conversores** que serán sustituidos con los valores de las variables convertidos al formato indicado.
- La salida es **bufereada**, lo que implica que sólo aparece en pantalla cuando existe un retorno de carro o al finalizar el programa.

Conversor	Variable	Salida	Conversor	Variable	Salida
%d %i	int short	int	%f	Float double	double (punto)
%u	int	Unsigned int	%e %E	Float double	double (exponencial)
%o, %x %X	int	Octal, hexadecimal	%g %G	Float double	double (conveniencia)
%c	char	char	%s %p	String puntero	string puntero

1. Entrada/Salida estándar: stdio.h

- También existen **modificadores** de los conversores especificando la anchura, la justificación o la precisión.

Significado	Modificador	Ejemplo
Justificación izquierda	-	%-d
Anchura de campo	número	%4f
.número	precisión	%.2f
tipo short	h	%hd
tipo long	l L	%ld %Lf

Ejemplos:

```
printf("el valor de entero es %d .\n", entero);        el valor de entero es 23 .
printf("%c%d\n", '$', entero);                         $23
printf("el valor de pi es %.2f\n",pi);                 el valor de pi es 3.14

scanf ("%d %f", &edad, &sueldo);                       // se pasan por referencia - direccion
scanf ("%s", tira);
23 21312.21 una_tira_de_caracteres                     ( o cada uno en una línea)
```

1. Entrada / Salida caracteres

- A la hora de dar los datos desde teclado, se pueden usar los caracteres separadores: retornos de carro, espacios, tabuladores
- El retorno de carro se quedará en el búfer de entrada, lo cual suele dar problemas a la hora de mezclar las entradas de enteros y caracteres.
- Hay otras dos llamadas para caracteres:
  - getchar() y putchar().
- Ejemplo:

```
#include <stdio.h>
int main()
{
    char ch;
    ch = getchar();
    putchar(ch);
    return 0;
}
```

¡Consejo! No poner espacios ni retornos de carro al final de la tira de control de scanf.

```
do {
   printf("Lo que queda en el
          buffer (ascii)\n");
   control=scanf("%c", &op);
   /* ¡!  op=getchar(); */
   printf("%d\n",op);
} while (op!=EOF);
```

Ejercicio 5: Entrada y Salida

1. Bájate el programa 6.c y guárdalo en tu directorio.
2. Ábrelo con un editor y examínalo y comprende las operaciones que hay.
3. Piensa como aparecerán en pantalla lo indicado en los tres primeros printf.
4. Introduce valores desde teclado para los tres siguientes scanf y piensa que valores están dando a las variables.
5. Ensaya dando los valores en una línea o en varias y dejando espacios o no.
6. ¿Qué ocurre con el último scanf?
7. ¿Qué valor toma el ultimo carácter con getchar?