Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

Tema 3: Introducción al Lenguaje C
OBJETIVOS

- ✓ Introducir los antecedentes históricos del lenguaje de programación C
- ✓ Mostrar la estructura general de un programa en C
- ✓ Detallar los distintos elementos que pueden aparecer en un programa en C: identificadores, palabras reservadas, literales, operadores y separadores
- ✓ Explicar los tipos de datos fundamentales del lenguaje, así como los distintos operadores que se pueden utilizar con ellos
- ✓ Introducir las operaciones básicas de lectura por teclado y escritura en pantalla

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

HISTORIA DE C / C++

El desarrollo inicial de C se produjo entre 1969 y 1973 (según Dennis Ritchie, el período más creativo fue 1972). Se llamó "C" porque muchas de sus características derivaban de un lenguaje anterior llamado "B", escrito por Ken Thompson **en 1970 para el primer sistema UNIX de la DEC PDP-7. El origen de B viene de BCPL, un lenguaje anterior escrito por Martin Richards para escribir sistemas operativos y compiladores. B y BCPL son lenguajes "carentes de tipos", pero C proporciona una variedad de tipos de datos.**

El lenguaje C fue una evolución del B llevada a cabo por Dennis Ritchie **en los Laboratorios Bell de California y originalmente se implementó en una computadora DEC PDP-11 en 1972. En 1973, C se había convertido en un lenguaje lo suficientemente poderoso como para reimplementar el núcleo de UNIX, probablemente siguiendo ejemplos del sistema multitarea MULTICS, implementado en PL/I, Tripos y posiblemente otros lenguajes.**

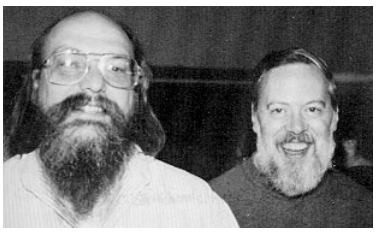
En 1978, Ritchie y Brian Kernighan **publicaron *El Lenguaje de Programación C*. A finales de los 70, C empezó a reemplazar a BASIC como lenguaje de los microcomputadores, siendo adoptado a la larga por el IBM PC.**

Un estudio de una distribución de Linux encontró que el 71% de sus 30 millones de líneas de código estaba escrito en C.

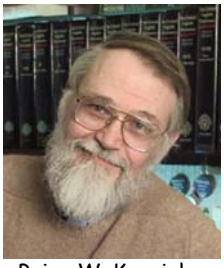
La popularidad de C aumento significativamente en los 80, lo que provocó que aparecieran demasiadas variantes del lenguaje. En 1989 fue oficialmente estandarizado por el ANSI y la ISO.

A finales de los 80, Bjarne Stroustrup y otros colegas de los Laboratorios Bell trabajaron para añadir construcciones de lenguajes orientados al objeto a C. El lenguaje que produjeron lo llamaron C++, ya que básicamente era una extensión de C.

C++ es un lenguaje híbrido, es posible programar en estilo imperativo (como en C), o en estilo orientado al objeto (como Java).



Ken Thompson Dennis Ritchie



Brian W. Kernighan

Bjarne Stroustrup

CARACTERÍSTICAS DEL C

- Es un lenguaje imperativo y estructurado
  - permite el uso de subrutinas y estructuras de control
- Es un lenguaje amigable, flexible y muy potente para el programador
  - C combina la flexibilidad de los lenguajes de alto nivel con el control y la funcionalidad que ofrecen lenguajes ensambladores (manipulación de bits, bytes y direcciones)
- Es un lenguaje eficiente
- Es un lenguaje portable
  - un programa escrito en ANSI C puede ejecutarse en cualquier ordenador con prácticamente ninguna modificación
- Es un lenguaje compilado

CONCEPTOS BÁSICOS

- Un dato es un elemento de información que puede ser
  - constante: no varía su valor (25, “hola”, numero_pi, ...)
  - variable: puede variar su valor (fecha, peso, altura, ...)
- Además, un dato puede:
  - ser literal (y por tanto constante) : 25, “hola”, 3.1415, ...
  - tener nombre (identificador): fecha, peso, altura, ...
- Un tipo de datos determina el rango de valores y el conjunto de operaciones posibles con dichos datos
- Ejemplo: si edad es un dato variable y edad_jubilación es un dato constante ambos del tipo números naturales, quedaría determinado:
  - edad puede tomar distintos valores del rango entero [0, ∞)
  - edad_jubilación sólo puede tomar un único valor de ese rango
  - las operaciones posibles con ambos datos serían: `+`, `-`, `*`, `/`,`%`

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

3.1. Introducción.
- 3.1.1. Historia de C / C++.
- 3.1.2 . Conceptos básicos: datos, tipos, variables y constantes.

3.2. INTEGRANTES DE C: el compilador, la librería estándar y el preprocesador.

3.3. Estructura general de un programa. La función **`main()`**.

3.4. Componentes sintácticos en C: tokens.

3.5. Tipos de datos fundamentales: **`int`**, **`float`**, **`char`** y conversiones.

3.6. Operadores, expresiones y sentencias.
- 3.6.1. Operadores: aritméticos, de asignación, incrementales, relacionales, lógicos, y otros.
- 3.6.2. Expresiones aritméticas, lógicas y generales.
- 3.6.3. Reglas de precedencia y asociatividad.
- 3.6.4. Sentencias: simples, compuestas, nulas.

3.7. Operaciones básicas de Entrada / Salida.

INTEGRANTES DEL LENGUAJE C

- El lenguaje C está constituido por tres elementos: el compilador, la librería estándar y el preprocesador:
  - El compilador:
    - Traduce a lenguaje máquina el programa escrito en C contenido en uno o más archivos fuente
    - Es capaz de detectar errores (fatales, de sintaxis o advertencias), indicando los correspondientes mensajes. En caso de existir sólo advertencias (warnings) no se impide la compilación
    - Los compiladores de C modernos vienen integrados en entornos visuales que hacen más manejable todo el proceso de edición, compilación, depuración y ejecución

INTEGRANTES DEL LENGUAJE C

- La librería estándar:
  - Conjunto de librerías con código objeto correspondiente a funciones preprogramadas que vienen junto con el compilador
  - Con objeto de mantener el lenguaje C lo más simple y sencillo posible muchas funciones que forman parte de otros lenguajes de programación, no tienen su correspondiente contrapartida en C. Por ejemplo las funciones de lectura / escritura
  - Sin embargo, esta funcionalidad tenía que ser cubierta por lo que se agrupan todas estas funcionalidades necesarias en un conjunto de librerías con código
  - Este conjunto de librerías es la librería estándar, que viene junto con el compilador
  - Se irán mostrando a lo largo del curso: stdio.h, string.h, math.h, ...

INTEGRANTES DEL LENGUAJE C

- El preprocesador:
  - Es un componente característico de C que no existe en otros lenguajes de programación. Actúa sobre el código fuente, antes de que empiece la compilación propiamente dicha
  - Las acciones a realizar vienen definidas por las directivas, que empiezan por almohadilla (#). Las directivas del preprocesador no son sentencias propiamente dichas (no terminan en punto y coma), simplemente avisan al compilador para realizar alguna acción previa a la compilación
  - Las directivas existentes son las siguientes:

    `#define, #undef, #if, #ifdef, #ifndef, #endif, #else, #elif, #include,#pragma,#error`
  - Las más utilizadas son `#include` y `#define`
  - Las directivas suelen aparecer al principio del programa, aunque no es obligatorio

INTEGRANTES DEL LENGUAJE C

- `#include`:

```
#include <nombre_archivo.h>
#include “nombre_archivo.h”
```

  - Indica al compilador que inserte, en el código fuente del programa, un archivo cabecera justo en la posición donde se encuentra la directiva
  - Si el nombre del fichero va entre ángulos (<>) el compilador busca el archivo cabecera en la librería estándar
  - Si el nombre del fichero va entre comillas (“”) el compilador busca el archivo cabecera en el directorio actual (donde se encuentra el código fuente)
  - Ejemplo: las funciones típicas de entrada / salida se encuentran en la librería estándar, en el archivo cabecera “stdio.h”. Si se quieren usar en un programa tiene que aparecer obligatoriamente la directiva `#include <stdio.h>`

INTEGRANTES DEL LENGUAJE C

- `#define`:

```
#define NOMBRE texto_sin_comillas
```

  - Establece una macro en el código fuente. Sustituye `NOMBRE` por el texto definido en la directiva
  - Ejemplo:

```
#define PI 3.1415
...
area = PI * r * r;
```

    Sustituye todas las apariciones de `PI` en el programa, por el texto definido en la directiva
  - Será el mecanismo que utilizaremos para definir constantes simbólicas en los programas y, como criterio, para distinguirlas rápidamente en los programas, usaremos las mayúsculas

INTEGRANTES DEL LENGUAJE C

```
#define NOMBRE(parámetros) texto_con_parámetros
```

- Establece una macro con parámetros en el código fuente
- Ejemplo:

```
#define PI 3.1415
#define CUAD(X) (X*X)
...
area = PI * CUAD(r);
```

Sustituye `CUAD(r)` por `(r * r)`

- En el preprocesamiento no se realiza ninguna revisión de tipos, ni de sintaxis, sólo se realizan sustituciones de código
- No usaremos las macros parametrizadas, sólo usaremos las macros que nos permitan definir constantes simbólicas

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

ESTRUCTURA DE UN PROGRAMA EN C

```c
#include <stdio.h>
#include <string.h>

 struct persona{
    char nombre[50];
    long telefono;
 };

void mostrar_persona(struct persona p);
struct persona leer_persona();

int main()
{
 struct persona p1;
 p1 = leer_persona();
 mostrar_persona( p1 );
 return 0;
}

void mostrar_persona(struct persona p )
{
 printf("\nNombre: %s", p.nombre);
 printf("\nTeléfono: %ld", p.telefono);
}
struct persona leer_persona()
{
 struct persona temp;
 printf("Nombre? ");
 gets(temp.nombre);
 printf("Telefono? ");
 scanf("%ld", &temp.telefono);
 return temp;
}
```

Directivas del preprocesador

Declaración de tipos

Declaración de funciones (prototipos)

La función `main()`

Definición de funciones creadas por el programador

Todo programa en C debe tener como mínimo:

- la función `main()`
- directivas del proprocesador (sólo si se necesita la librería estándar)

LA FUNCIÓN `main()`

- Todo programa en C, desde el más pequeño hasta el más complejo tiene como mínimo una función: la función `main()`
- Al ejecutar un programa compilado, se comienza por la función `main()` (por tanto habrá únicamente una en cada programa)
- Formato:

```c
int main()
 {
    sentencia_1;
    ...
    sentencia_n;
    return 0 ;
 }
```

Como se verá, debe existir correspondencia entre el tipo de devolución de la función y el valor que se devuelve

- El bloque de sentencias va incluido entre llaves (`{...}`)
- Dentro de la función puede haber llamadas a otras funciones: propias o de la librería estándar
- Se termina la ejecución del programa al llegar a la sentencia `return 0`. Indica que la función ha terminado correctamente (devuelve el valor cero)

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

3.1. Introducción.
- 3.1.1. Historia de C / C++.
- 3.1.2 . Conceptos básicos: datos, tipos, variables y constantes.

3.2. Integrantes de C: el compilador, la librería estándar y el preprocesador.

3.3. Estructura general de un programa. La función **`main()`**.

3.4. COMPONENTES SINTÁCTICOS EN C: TOKENS.

3.5. Tipos de datos fundamentales: **`int`**, **`float`**, **`char`** y conversiones.

3.6. Operadores, expresiones y sentencias.
- 3.6.1. Operadores: aritméticos, de asignación, incrementales, relacionales, lógicos, y otros.
- 3.6.2. Expresiones aritméticas, lógicas y generales.
- 3.6.3. Reglas de precedencia y asociatividad.
- 3.6.4. Sentencias: simples, compuestas, nulas.

3.7. Operaciones básicas de Entrada / Salida.

COMPONENTES SINTÁCTICOS EN C

- Los compiladores descomponen los programas o códigos fuente en componentes sintácticos (o tokens) y a partir de esta descomposición generan el código objeto correspondiente
- En el lenguaje de programación C existen cinco tipos de tokens: identificadores, palabras reservadas, constantes, operadores y separadores:
  - Identificadores:
    - Un identificador es un nombre con el que se hace referencia a una función o al contenido de una zona de memoria (constantes o variables)
    - En ANSI C, el nombre de un identificador debe ser una secuencia de exclusiva de: letras del alfabeto (a..z, A..Z), caracteres subrayado (_) y dígitos numéricos (0..9). No puede empezar por dígito numérico y se distinguen mayúsculas de minúsculas (elem_mayor, var1, PI, _edad, Cont_12, ...)

COMPONENTES SINTÁCTICOS EN C

- Palabras reservadas:
  - Todos los lenguajes de programación tienen un conjunto de palabras reservadas (*keywords*) que tienen un significado especial dentro del lenguaje, por lo que no se pueden utilizar como nombre de identificadores
  - ANSI C tiene un conjunto de 33 palabras reservadas:

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

  - Se irá mostrando su uso y significado a lo largo del curso

COMPONENTES SINTÁCTICOS EN C

- Constantes:
  - Existen dos tipos de constantes: literales (numéricas, carácter, y cadena de caracteres) y simbólicas:
    - numéricas: `25`, `3.5`, `4e5`, `2.4e10`, ...
    - carácter: `'a'`, `'A'`, `'*'`, `'?'`, `'%'`, ... (ASCII)
    - cadena de caracteres: `"Buenos dias, señor"` ...
    - simbólicas: tienen asociado un identificador y se definen en las sentencias del preprocesador (`#define PI 3.1415`)
- Operadores:
  - Signos especiales que indican distintas operaciones a realizar con las variables y/o constantes del programa

```
! % ^ & * ( ) - + = { } ~
[ ] \ ; ' : < > ? , . / "
```

COMPONENTES SINTÁCTICOS EN C

- Separadores:
  - Se considera separador a uno o más espacios en blanco, tabuladores, y caracteres nueva línea
  - En si mismos no son componentes sintácticos, ayudan al compilador a descomponer el programa fuente en cada uno de sus tokens
  - Es conveniente utilizarlos incluso sin ser necesarios, con objeto de mejorar la legibilidad de los programas
  - A todos los efectos los comentarios se consideran separadores puesto que son ignorados por el compilador
  - Se considera comentario a todos los caracteres comprendidos entre `/*` y `*/` (incluyendo estas marcas), pudiendo ocupar varias líneas y existir varios de ellos a lo largo de un programa
  - Los comentarios aumentan la legibilidad de los programas indicando una explicación (comentario) a alguna operación, función, variable ...

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

TIPOS DE DATOS FUNDAMENTALESy

- C, como cualquier otro lenguaje de programación, tiene posibilidad de trabajar con datos de distinta naturaleza: números enteros (`int`), número reales (`float`), caracteres (`char`) ...
- Además, algunos de estos tipos de datos admiten distintos números de cifras (rango y/o precisión) y posibilidad de ser sólo positivos o de ser positivos y negativos

modo abreviado

caracteres	**`char`**		
números naturales	**`unsigned short`** `int`	**`unsigned`** `int`	**`unsigned long`** `int`
números enteros	**`short`** `int`	**`int`**	**`long`** `int`
números reales	**`float`**	**`double`**	**`long double`**

tipos de datos simples fundamentales de C

TIPOS DE DATOS FUNDAMENTALES

- Números enteros: el tipo **`int`**
  - Representa a los números enteros con signo
  - Rango de valores
    - Depende del compilador que se use, aunque normalmente ocupa 2 bytes (16 bits)

$$[\ -2^{15},\ 2^{15}-1\ ] = [\ -32.768\ ,\ 32.767\ ]$$

  - Modificadores
    - **`short`** (**`int`**): normalmente ocupa 1 byte (8 bits) por lo que su rango de representación es

$$[\ -2^{7},\ 2^{7}-1\ ] = [\ -128\ ,\ 127\ ]$$

    - **`long`** (**`int`**): normalmente ocupa 4 bytes (32 bits) por lo que su rango de representación es

$$[\ -2^{31},\ 2^{31}-1\ ] = [\ -2.147.483.648,\ 2.147.483.647\ ]$$

TIPOS DE DATOS FUNDAMENTALES

- Números naturales: el tipo **`unsigned (int)`**
  - Representa a los números enteros sin signo
  - No es un tipo propiamente dicho, es un modificador de `int`
  - Rango de valores: igual que el tipo `int`, pero sin simetría en el intervalo puesto que no se consideran los números negativos:

$$[\ 0,\ 2^{16}-1\ ] = [\ 0\ ,\ 65.535\ ]$$

  - También se pueden usar los modificadores `short` y `long`:
    - **`unsigned short (int)`**:

$$[\ 0,\ 2^{8}-1\ ] = [\ 0\ ,\ 255\ ]$$

    - **`unsigned long (int)`**:

$$[\ 0,\ 2^{32}-1\ ] = [\ 0\ ,\ 4.294.967.295\ ]$$

TIPOS DE DATOS FUNDAMENTALES

- Declaración de variables enteras
  - Se pone el nombre del tipo seguido del identificador de la variable (o varias variables separadas por comas):

```
nombre_tipo identificador_variable;
```

  - Ejemplos:

```
short numero_menor;
int num1, num2, num3;
long numero_mayor;
unsigned short contador;
unsigned numero_natural;
unsigned long num_dni;
```

se pueden declarar varias variables del mismo tipo en la misma sentencia separándolas con comas

  - Se puede dar valor a la variable a la vez que se declara:

```
int numero = -345;
```

TIPOS DE DATOS FUNDAMENTALES

- Constantes literales enteras
  - No se pueden utilizar comas, puntos, ni cualquier otro signo de puntuación que se utiliza normalmente para representar cantidades en contabilidad:

    `123456` en vez de `123.456`

  - Por defecto una constante entera literal es de tipo **`int`**, o **`long`** si se sale del rango
  - Se pueden utilizar sufijos para forzar el tipo de la constante literal **`U`** y/o **`L`** (también en minúscula):

`23484`	constante tipo `int`
`45815`	constante tipo `long` (mayor que 32767)
`253u` ó `253U`	constante tipo `unsigned int`
`739l` ó `739L`	constante tipo `long`
`583ul` ó `583UL`	constante tipo `unsigned long`

TIPOS DE DATOS FUNDAMENTALES

- En C se puede expresar una constante entera octal, esto es, expresada en base 8 (dígitos del 0 al 7). Para ello simplemente se precede la constante con un cero (0):

`011`	constante octal (9 en base 10)
`11`	constante entera decimal (no es igual a 011)

- Análogamente, una secuencia de dígitos (del 0 al 9) y de letras (A..F) precedida por `0x` ó `0X`, se interpreta como una constante entera hexadecimal, esto es, expresada en base 16:

`0xB`	constante hexadecimal (11 en base 10)
`0xFF`	constante hexadecimal (255 en base 10)

TIPOS DE DATOS FUNDAMENTALES

- Caracteres: el tipo **`char`**
  - Representa a los caracteres
  - Internamente no almacena el carácter, almacena el valor numérico de su posición en el código ASCII (ver tema 1)
  - Rango de valores
    - Ocupa un byte (8 bits)

$$[\ 0,\ 2^8\text{-}1\ ] = [\ 0,\ 255\ ]$$

  - Declaración de variables:

```
char vocal;
char letra1,letra2;
```


TIPOS DE DATOS FUNDAMENTALES

- Las constantes literales de tipo carácter van entre comillas simples: `'A'`, `'a'`, `'8'`, `'$'`, ...
- Existen dos formas de asignar un valor directamente a una variable de tipo carácter:
  - Directamente el carácter que se quiera:

```
char vocal = 'A';
```

  - Asignándole el valor numérico de su posición en el código ASCII:

```
char letra = 80;
/* asigno la letra 'p' */
```

- De la misma forma se puede hacer:

```
char c = 'J';
c = c + 1;          /* c toma el valor 'K' */
c = c + 'a' - 'A' /* la paso a minúscula 'k'*/
```

TIPOS DE DATOS FUNDAMENTALES

- Existen caracteres especiales (algunos de ellos no son imprimibles) que se representan precedidos de \ . Los más usuales son:

Carácter	ASCII	Significado
\n	10	nueva línea
\t	9	tabulador
\b	8	retroceso
\r	13	retorno de carro
\f	12	avance de página
\\	92	barra inclinada inversa
\'	39	comilla simple
\"	34	doble comillas
\a	7	alerta (pitido)
\0	0	carácter nulo (fin de cadena)

- Estos caracteres se asignan igual que los demás:

```
char c = 10;
/* es lo mismo que c = '\n'; */
```


TIPOS DE DATOS FUNDAMENTALES

- Números reales (con coma flotante): el tipo **`float`**
  - Representa a los números reales, tales como 3.1415 ó $1.85 \cdot 10^{15}$
  - Rango de valores: depende del compilador que se use y del modificador:
    - **`float`**: normalmente ocupa 4 bytes (32 bits). Su rango de representación es

      $[\ 3.4 \cdot 10^{-38}, 3.4 \cdot 10^{38}\ ]$ con 7 dígitos de precisión
    - **`double`**: normalmente ocupa 8 bytes (64 bits). Su rango de representación es

      $[\ 1.7 \cdot 10^{-308}, 1.7 \cdot 10^{308}\ ]$ con 15 dígitos de precisión
    - **`long double`**: normalmente ocupa 10 bytes (80 bits). Su rango de representación es

      $[\ 3.4 \cdot 10^{-4932}, 3.4 \cdot 10^{4932}\ ]$ con 19 dígitos de precisión

TIPOS DE DATOS FUNDAMENTALES

- Declaración de variables punto flotante

```
float peso, altura, numero_real;
double numero_real_mayor;
long double numero_real_mucho_mas grande;
```

- Constantes literales en punto flotante
  - Por defecto, las constantes literales en punto flotante se consideran de tipo **`double`**. Si aparecen terminadas con **`f`** o **`F`** se cosideran de tipo **`float`** y terminadas con **`l`** o **`L`** de tipo **`long double`**

`3.53 , -15.64`	constantes de tipo `double`
`3.53f , -15.64F`	constantes de tipo `float`
`3.53l , -15.64L`	constantes de tipo `long double`

TIPOS DE DATOS FUNDAMENTALES

- Puede utilizarse también la notación científica: la constante tendrá una parte entera, un punto decimal, una parte fraccionaria, una **`e`** o **`E`**, y un exponente entero (pudiendo ser negativo)
- Se puede omitir la parte entera o la fraccionaria, pero no ambas a la vez
- Ejemplos:

`.874e-2`	constante de tipo `double ( = 0.00874)`
`.874e-2f`	constante de tipo `float`
`5.2E3`	constante de tipo `double ( = 5200.0)`
`1,23`	incorrecta: la coma no sirve de punto decimal
`23963f`	incorrecta: no hay punto decimal
`.e4`	incorrecta: no hay parte entera ni fraccionaria

TIPOS DE DATOS FUNDAMENTALES

- El tipo lógico
  - Los compiladores que siguen el estándar ANSI no incorporan el tipo lógico: se utiliza el tipo entero para representarlo
  - En el lenguaje C, una expresión es falsa si es evaluada a cero (`0`) y verdadera si es evaluada a uno (`1`)
  - En realidad, cualquier valor distinto de cero se evalúa como verdadero (lo veremos en el apartado de expresiones lógicas)
- El tipo nulo: **`void`**
  - Se utiliza para indicar (si es el caso) que una función no devuelve ningún valor o para indicar que una función no tiene argumentos (esto último es opcional)
  - Lo mostraremos en el Tema 5 (subprogramas)

CONVERSIONES DE TIPOS

- Conversiones implícitas: cuando en una expresión se mezclan constantes y variables de distintos tipos, el compilador realiza conversiones automáticas de tipos siguiendo las siguientes reglas:
  - En cualquier operación en la que aparezcan dos tipos diferentes se eleva el rango del menor para igualarlo al del mayor.

    A esta conversión, en la que el programador no interviene, se le denomina promoción, pues la variable de menor rango es *promocionada* a la de mayor rango.

    Los tipos de mayor a menor rango son:

    ```
    long double > double > float > unsigned
    long > long > unsigned int > int > char
    ```

CONVERSIONES DE TIPOS

- En una sentencia de asignación, el resultado final de los cálculos se reconvierte al tipo de la variable a la que está siendo asignada. El proceso puede ser una promoción o una pérdida de rango según la categoría de la variable a la que se le efectúa la asignación

- Conversiones explícitas: se denominan también castings. Ocurre cuando el programador fuerza la conversión de un tipo a otro de forma explícita.

  La forma de realizar un casting es poniendo el tipo deseado entre paréntesis delante de la expresión que se desea convertir

```
int   x, j = 2;
float   y =  1.0, z = 2.0;
z = y + j;       /* promoción de j */
x = y + z;       /* pérdida de rango */
x = (int) y + (int) z;    /* casting */
```


Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

OPERADORES

- Un operador es un carácter, o grupo de caracteres, que actúa sobre una, dos o más variables para realizar una determinada operación con un determinado resultado
- Los operadores pueden ser unarios o binarios, según actúen sobre uno o dos operandos respectivamente
- En C existen diversos operadores (éste es uno de los puntos fuertes del lenguaje), que veremos a continuación:
  - Aritméticos
  - De asignación
  - Incrementales
  - Relacionales
  - Lógicos
  - Otros

OPERADORES ARITMÉTICOSy

- Los operadores aritméticos realizan las operaciones aritméticas básicas
- En C son todos binarios y se pueden aplicar a variables, constantes y expresiones:
  - `+` suma
  - `-` resta
  - `*` multiplicación
  - `/` división
  - `%` resto
- El operador `/`, si los dos operandos son enteros, obtiene la división entera (sin decimales) de los mismos
- El operador `%` sólo tiene sentido entre operandos enteros, obteniendo como resultado el resto de la división de los mismos

OPERADORES DE ASIGNACIÓN

- Los operadores de asignación son operadores binarios que asignan a una variable (situada a la izqda. del operador) el valor resultante de evaluar una expresión (situada a la derecha del operador)
- El operador de asignación habitual en todos los lenguajes de programación es el símbolo igual (=)

```
nombre_de_variable = expresión;

resultado = (x + y – z) / 4;
mi_var = 5;
```

- El valor que tuviese la variable antes de la asignación se pierde
- El resultado de la expresión debe ser del mismo tipo que la variable, en caso contrario se puede producir una pérdida de rango en conversiones implícitas de tipos

OPERADORES DE ASIGNACIÓN

- No hay que confundir la asignación con la igualdad matemática. La siguiente expresión no tendría sentido desde el punto de vista matemático:

```
variable = variable + 1;
```

  Simplemente suma uno al valor de la variable
- A la izquierda de un operador de asignación no puede haber expresiones, sólo puede aparecer el nombre de una variable

```
a + b = c; /* error */
```

- En C es posible la asignación múltiple y simultánea de varias variables a un mismo valor:

```
a = b = c = d = expresión;
```

OPERADORES DE ASIGNACIÓN

- En C existen otros cuatro operadores de asignación (`+=`, `-=`, `*=`, `/=`) que simplifican algunas operaciones recurrentes sobre una misma variable
- Su forma de uso es

```
variable op= expresión; /* op ⊂ {+,-,*,/} */
```

que es totalmente equivalente a

```
variable = variable op expresión;
```

- Ejemplos:

```
cont += 1;          /* cont = cont + 1;     */
rango /= 2.0;       /* rango = rango /2.0; */
x *= 3.0 + y;       /* x = x * (3.0 + y);   */
```


OPERADORES INCREMENTALES

- Los operadores incrementales son operadores unarios que incrementan (`++`) o decrementan (`--`) en una unidad el valor de la variable a la que afectan
- Pueden ir inmediatamente delante (pre) o inmediatamente detrás (post) de la variable a la que afectan
  - Preincremento, Predecremento
    - La operación de incremento o decremento se lleva a cabo antes de utilizar el valor del operando, es decir, primero se incrementa (o decrementa) el valor del operando y luego se utiliza
  - Postincremento, Postdecremento
    - La operación de incremento o decremento se lleva a cabo después de utilizar el valor del operando, es decir, primero se utiliza el valor del operando y luego se incrementa (o decrementa)

OPERADORES INCREMENTALES

- Ejemplos:

Operación	Resultado
`x++;`	`x = x + 1;`
`++x;`	`x = x + 1;`
`--x;`	`x = x – 1;`
`x = 100;` `y = ++x;`	`x = 101;` `y = 101;`
`x = 100;` `y = x++;`	`x = 101;` `y = 100;`

`x = 5;`
`y = 2;`
`++x += --y;`
	`x = 7;`
	`y = 1;`

OPERADORES RELACIONALES

- Una característica imprescindible de cualquier lenguaje de programación es poder variar el flujo del programa según se cumplan o no ciertas condiciones
- Los operadores relacionales permiten hacer comparaciones, obteniendo como resultado verdadero (`1` ó `≠0`) o falso (`0`) dependiendo si se cumplen o no ciertas condiciones
- En C, los operadores relacionales son todos binarios:
  - `==` igual que
  - `<` menor que
  - `>` mayor que
  - `<=` menor o igual que
  - `>=` mayor o igual que
  - `!=` distinto que

```
expresión op expresión
```

OPERADORES LÓGICOS

- Los operadores lógicos permiten combinar los resultados de los operadores relacionales, comprobando si se cumplen simultáneamente o no varias condiciones
- Formato:

```
expresión op_lógico expresión
```

- En C existen tres operadores lógicos:
  - && conjunción lógica (binario): el resultado es verdadero si ambas expresiones a las que afecta son verdaderas
  - || disyunción lógica (binario): el resultado es verdadero si alguna de las expresiones a las que afecta es verdadera
  - ! negación lógica (unario): el resultado es verdadero si la expresión a la que afecta es falsa y viceversa

OPERADORES LÓGICOS

- Tablas de verdad:

Operador	Acción
&&	Y
||	O
!	NO

a	b	a && b	a || b	! a	! b
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

OTROS OPERADORES

- `-` operador menos (unario): cambia el signo de la variable o expresión que tenga asociada. En C no hay constantes literales reales negativas.

  Formato:

```
- expresión
```

- `sizeof()` operador tamaño (unario): obtiene el tamaño, en bytes, de la variable o tipo que tenga entre los paréntesis. Recuérdese que este tamaño depende del compilador, por lo que es necesario disponer de este operador para producir código portable.

  Formato:

```
sizeof(nombre_tipo)  o  sizeof(nombre_variable)
```

- `&` operador de dirección y `*` operador de indirección (unarios): relacionados con los punteros. Los veremos a lo largo del curso

EXPRESIONES

- Una expresión es una combinación de elementos que representan valores (operandos) y conectivas que representan operaciones (operadores)
- Operandos:
  - Constantes literales
  - Constantes simbólicas
  - Variables
  - Llamadas a funciones
- Operadores:
  - Aritméticos
  - Incrementales
  - Relacionales
  - Lógicos
- Una expresión puede estar formada por otras expresiones más sencillas y puede contener paréntesis que agrupen a los distintos términos

```
s <= ((d+4.0e-5)*32.1)/sqrt(PI*valor) && ! fin
```

EXPRESIONES

- Las expresiones son equivalentes al resultado que proporciona el aplicar los operadores a los operandos
  - Por ejemplo, `1+5` es una expresión formada por los operandos `1` y `5` y el operador `+`. La expresión es equivalente al valor `6`, por tanto allí donde esta expresión aparece en el programa, es evaluada y sustituida por su valor `6`
- Las expresiones deben tener tipos compatibles. En caso contrario, se producen conversiones implícitas que pueden producir pérdidas de rango y, por tanto, obtenerse resultados no válidos
- En C, como en la mayoría de los lenguajes de programación, existen expresiones aritméticas, expresiones lógicas y expresiones generales

EXPRESIONES ARITMÉTICAS

- Las expresiones aritméticas son aquellas en las que aparecen exclusivamente operadores aritméticos y/o incrementales (`+`, `-`, `*`, `/`, `++`, `--`) y paréntesis (en caso necesario)
- La resolución de las raíces de una ecuación de segundo grado de la forma $ax^2 + bx + c = 0$ sería:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

En C, se escribiría

```
(-b+sqrt((b*b) -(4*a*c)))/(2*a)
(-b-sqrt((b*b) -(4*a*c)))/(2*a)
```

EXPRESIONES LÓGICAS

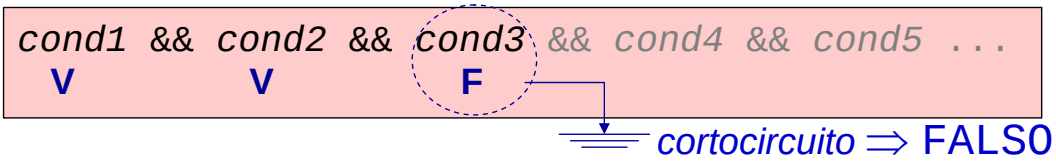
- Las expresiones lógicas son aquellas en las que aparecen exclusivamente operadores relacionales y/o lógicos (>, <, >=, <=, ==, !=, &&, ||, !) y paréntesis (en caso necesario)
- El resultado de una expresión lógica es verdadero (≠0) o falso (0)

```
es_primo
x >= 0
!((letra=='Q') || (letra=='z')) && (x>=6)
(largo==ancho) && !(x<5)
x && (altura<=5)
(m!=10) || (z>5)
```

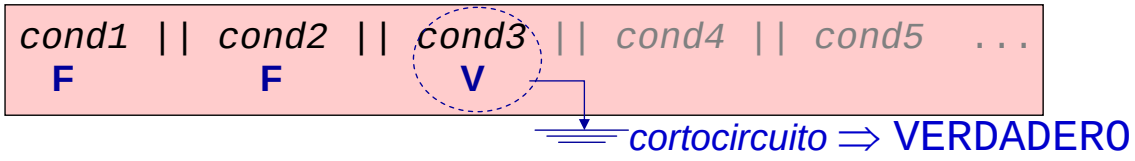
- Las expresiones lógicas se utilizan en las estructuras de control para permitir alterar el flujo del programa

EXPRESIONES LÓGICAS

- En el lenguaje C, como en otros lenguajes de programación, las expresiones lógicas se evalúan en cortocircuito: al evaluar un operando podría concluirse que toda la expresión lógica fuese verdadera o falsa
  - Cuando un operando de una conjunción lógica (&&) es falso, toda la expresión es falsa



  - Cuando un operando de una disyunción lógica (||) es verdadero, toda la expresión lógica es verdadera



EXPRESIONES LÓGICAS

- Leyes de Morgan:

```
!(cond1 && cond2 && cond3 && ... )
                ⇔
  !cond1 || !cond2 || !cond3 ...
```

```
!(cond1 || cond2 || cond3 || ... )
                ⇔
  !cond1 && !cond2 && !cond3 ...
```

```
!(b>=a && sqrt(x)>=1.0)
          ⇔
  (b<a) || sqrt(x)<1.0
```

```
!(a==b || ok || !(x<y))
          ⇔
  a!=b && !ok && (x<y)
```

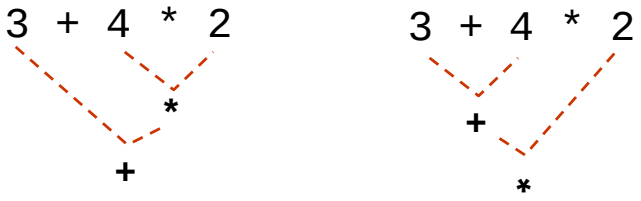

EXPRESIONES GENERALES

- Una de las características más importantes del C (y en ocasiones difíciles de interpretar) es su flexibilidad para combinar expresiones y operadores de distintos tipos en una expresión general
- Se acaba de mostrar que el resultado de una expresión lógica es siempre un valor numérico (`0` ó `≠0`). Esto permite que cualquier expresión lógica pueda aparecer como sub-expresión en una expresión aritmética y viceversa
- A su vez, el operador de asignación (`=`), además de asignar un valor a una variable, deja este valor disponible para ser utilizado en una expresión que lo englobe

```
!(a>3) && (b=c=d) || (8-f++)
```

REGLAS DE PRECEDENCIA Y ASOCIATIVIDAD

- El resultado de una expresión depende del orden en el que se evalúen las operaciones. El siguiente ejemplo ilustra claramente la importancia del orden:



- El orden de evaluación de las operaciones puede modificarse mediante paréntesis: se realizan primero las operaciones encerradas en los paréntesis interiores

```
3 + (4 * 2)
```

```
(3 + 4) * 2
```

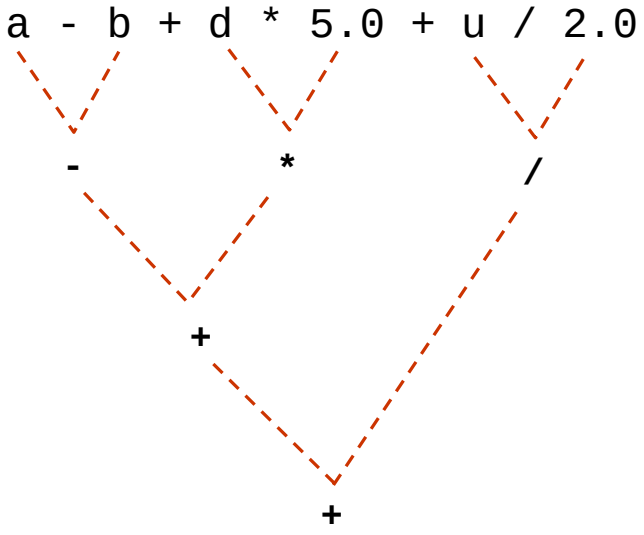
- No obstante, si no se utilizasen paréntesis, el resultado de toda expresión debe quedar claro e inequívoco. Por tanto, es necesario definir reglas que indiquen el orden con el que se ejecutan las expresiones en C: reglas de precedencia y de asociatividad

REGLAS DE PRECEDENCIA Y ASOCIATIVIDAD

- Reglas de precedencia y de asociatividad

PRECEDENCIA	ASOCIATIVIDAD
`()  []  ->  .`	izquierda a derecha
`++  --  !  sizeof`	derecha a izquierda
`-(unario)`	derecha a izquierda
`*(indirección)  &(dirección)`	derecha a izquierda
`*  /  %`	izquierda a derecha
`+  -`	izquierda a derecha
`<  <=  >  >=`	izquierda a derecha
`==  !=`	izquierda a derecha
`&&`	izquierda a derecha
`||`	izquierda a derecha
`?:`	izquierda a derecha
`=  +=  -=  *=  /=`	derecha a izquierda

REGLAS DE PRECEDENCIA Y ASOCIATIVIDAD



SENTENCIAS

- Las expresiones son unidades, o componentes elementales, de entidades de rango superior: las sentencias
- Una sentencia es una unidad completa y ejecutable en si misma que indica una acción a realizar
- Las sentencias pueden ser simples, compuestas o nulas:
  - Sentencia simple: ocupa una única línea del programa y va terminada en punto y coma (`;`)
  - Sentencia compuesta: es un conjunto de sentencias (simples y/o compuestas) agrupadas entre llaves (`{}`). Se utilizan dentro del cuerpo de las funciones y de las sentencias de control
  - Sentencia vacía o nula: en algunas ocasiones es necesario introducir en el programa una sentencia que ocupe un lugar, pero que no realice ninguna tarea. A esta sentencia se le denomina sentencia vacía y consta de un simple carácter punto y coma (`;`)

SENTENCIAS

```
/* Resolución de una Ecuación de 2ºgrado */

void ec2grado(float a, float b, float c)
 {
   float x1, x2, discr;
   discr = b*b-4.0*a*c;
   if (discr>=0.0)
     {
       x1 = (-b+sqrt(discr))/(2.0*a);
       x2 = (-b-sqrt(discr))/(2.0*a);
       printf("Soluciones: %f y %f", x1, x2);
     }
    else
       ;
 }
```

sentencias simples

sentencia compuesta

sentencia nula

Tema 3: Introducción al Lenguaje C
Estructura y Contenidos

OPERACIONES BÁSICAS DE E/S

- Las funciones de entrada/salida permiten a un programa recibir y enviar datos al exterior. Por defecto la lectura se hace por el teclado y la escritura en la pantalla del ordenador

- Como se ha comentado a lo largo del tema, las funciones de entrada/salida no pertenecen propiamente al lenguaje de programación C: están definidas en la librería estándar

- Para su utilización es necesario incluir, al comienzo del programa, el archivo **`stdio.h`** (o **`conio.h`**), que es donde están definidos los prototipos de dichas operaciones

- Vamos a explicar las operaciones básicas de lectura por teclado y escritura por pantalla (**`scanf()`**, **`printf()`**, **`getchar()`**, **`putchar()`**, **`getch()`**, **`getche()`**)

ESCRITURA POR PANTALLA

- La escritura de datos con formato se realiza con la función **`printf()`** cuyo formato es

```
printf("cadena_de_control",lista_de_argumentos);
```

  - **`cadena_de_control`**: es una cadena de texto que contiene el texto que se quiere escribir directamente por pantalla y unos caracteres de conversión que indicarán el formato con en el que se escribirán los argumentos de la **`lista_de_argumentos`**

  - **`lista_de_argumentos`**: es una lista de expresiones, variables o constantes, separadas por comas, que se escribirán en lugar de los caracteres de conversión de la **`cadena_de_control`**

- Tiene que haber el mismo número de caracteres de conversión (en la **`cadena_de_control`**) que de argumentos (en la **`lista_de_argumentos`**) y se asociarán en el mismo orden

ESCRITURA POR PANTALLA

Carácter de conversión	Tipo de argumento	Formato de salida
%d	entero	entero decimal con signo
%u	entero	entero decimal sin signo
%i	entero	entero decimal con signo
%x	entero	entero hexadecimal sin signo (letras minúsculas)
%X	entero	entero hexadecimal sin signo (letras mayúsculas)
%o	entero	entero octal sin signo
%f	real	[-] dddd.dddddd (en notación habitual)
%e	real	[-] d.ddde[+|-]ddd (en notación científica)
%E	real	[-] d.dddE[+|-]ddd (en notación científica)
%c	carácter	carácter del código ASCII
%s	cadena de texto	cadena de caracteres
%p	puntero	dirección de memoria

ESCRITURA POR PANTALLA

- Modificadores del formato: en los caracteres de conversión, entre el carácter % y el carácter que indica el tipo, puede aparecer en este orden:
  - un signo menos (**-**), que indica alineamiento a la izquierda (por defecto es a la derecha)
  - un número entero positivo, que indica la anchura mínima del campo en caracteres. Si hay un cero delante se rellena con ceros
  - un punto (**.**), que separa la anchura de la precisión
  - un número entero positivo, la precisión, que es el número máximo de caracteres a imprimir en una cadena de caracteres; el número de decimales de un **float** o **double**; las cifras mínimas de un **int** o **long**
  - un cualificador de tipo: una **h** para **short** o una **l** para **long** y **double**

ESCRITURA POR PANTALLA

```
#include <stdio.h>
int main()
 {
   int a = 2500;
   short b = 10;
   long c = 123456;
   float d = 134.123456;
   double e = 12345678.123456;
   printf("1. 123456789012345\n");
   printf("2. %d\n",a);
   printf("3. %010d\n",a);
   printf("4. %-10d*\n",a);
   printf("5. %hd\n",b);
   printf("6. %ld\n",c);
   printf("7. %f\n",d);
   printf("8. %10.2f\n",d);
   printf("9. %-10.2f\n",d);
   printf("10. %.3lf\n",e);
   printf("11. %015s\n","1234567");
   return 0;
 }
```

```
1. 123456789012345
2. 2500
3. 0000002500
4. 2500      *
5. 10
6. 123456
7. 134.123459
8.     134.12
9. 134.12
10. 12345678.123
11. 000000001234567
```


LECTURA POR TECLADO

- La lectura de datos con formato se realiza con la función `scanf()` cuyo formato es

```
scanf("%x1 %x2 ...%xn",&var1, &var2, ... &varn);
```

  - `%x1 %x2 ... %xn`: es una lista de caracteres de conversión que indica el número de valores y el tipo de los mismos que se quieren leer por teclado
  - `&var1, &var2, ... &varn`: es una lista de variables separadas por comas donde se almacenan los valores leídos por teclado. Las variables van precedidas, normalmente, del carácter & (más adelante detallaremos esta cuestión)

- Tiene que haber el mismo número de caracteres de conversión que de variables
- A la hora de la lectura, dos elementos se consideran distintos si van separados por un espacio en blanco, tabulador o salto de línea

LECTURA POR TECLADO

```c
/* Área de un rectángulo */
#include <stdio.h>
int main()
 {
   float base, altura, area;
   printf("Introduce la base y la altura: ");
   scanf("%f %f",&base,&altura);
   area = base * altura;
   printf("\nEl area es %.2f",area);
   return 0;
 }
```

```
Introduce la base y la altura: 3.6  2.5
El area es 9.00
```


LECTURA/ ESCRITURA DE UN CARÁCTER

- Existen otras funciones de lectura y escritura de un solo carácter que se usan habitualmente:
  - ```c
    putchar(letra);      /* totalmente equivalente a
                            printf("%c",letra);    */
    ```
  - ```c
    letra = getchar();   /* totalmente equivalente a
                            scanf("%c",&letra);    */
    ```
  - ```c
    letra = getche();    /* lee un carácter sin pulsar
                            intro                  */
    ```
  - ```c
    letra = getch();     /* igual que getche() pero
                            no hace eco en pantalla */
    ```
- Las dos últimas (`<conio.h>`) pueden utilizarse a modo de pausa sin necesidad de asignar el carácter leído a ninguna variable

EJEMPLOS

Escribe un programa que lea dos valores enteros por teclado almacenándolos en variables distintas. Debe Intercambiar los valores de ambas variables y mostrarlos por pantalla:

a) usando una variable auxiliar

b) sin usar variables auxiliares

Escribe un programa que lea por teclado el radio de una circunferencia y muestre por pantalla su área y perímetro.

Utiliza una constante simbólica para el número π

Escribe un programa que lea una cantidad, que representa un número de segundos, e indique a cuantas horas, minutos y segundos corresponde.

P.e. 3723 segundos corresponden a 1 hora, 2 minutos y 3 segundos

Tema 3: Introducción al Lenguaje C

FIN DEL TEMA