Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

TEMA 3: Introducción a C

Fundamentos de Informática (Ingeniería Técnica Industrial)

E.U. Politécnica

Autores: M.C.Aranda, A.Fernández, J.Galindo, M.Trella

Índice de contenidos

• *Introducción al lenguaje C.*

- El lenguaje de programación C fue creado por Dennis Ritchie en 1972 usando UNIX como Sistema Operativo.
- El C deriva del lenguaje B de Ken Thompson.
- La rápida expansión de C sobre varios tipos de computadoras hizo que surgieran muchas variantes =>
  - En 1989 se unificaron criterios y se adoptó el estándar ANSI de C conocido como ANSI C o C89.
  - En 1999 se desarrolló otro estándar conocido como el C99
- Características de C:
  - Es un lenguaje estructurado
    - Permite el uso de subrutinas y estructuras de control
  - Es un lenguaje amigable, flexible y muy potente para el programador
    - El C combina elementos de lenguaje de alto nivel con el control y la flexibilidad que ofrecen lenguajes ensambladores (manipulación de bits, bytes y direcciones).
  - Es un lenguaje eficiente
  - Es un lenguaje portátil
    - Un programa escrito en C puede ejecutarse en cualquier ordenador con muy pocas modificaciones o ninguna.
  - Es un lenguaje compilado

• *Compilación.*

- Para realizar un programa en C hay que seguir los siguientes pasos:
  - Diseñar el algoritmo que resuelve el problema que estamos tratando de solucionar.
  - Escribir el **código fuente** del programa *(nombre_programa.c):*
    - "Traducir" el algoritmo escrito en pseudocódigo o diagramas de flujo al lenguaje de programación que vayamos a usar.
    - Esta tarea se puede realizar utilizando cualquier editor de texto.
  - Generar el **código objeto** del programa *(nombre_programa.obj):*
    - Se ejecuta un programa llamado **compilador** indicándole cual es nuestro código fuente *(nombre_programa.c).*
    - El compilador traduce el programa fuente al lenguaje interno del ordenador.
    - El compilador comprueba si hemos cometido algún error de sintaxis al escribir el código fuente.
      - Si hay errores => se corrigen y se vuelve a ejecutar el compilador
  - Generar el **fichero ejecutable** *(nombre_programa.exe)*
    - Se ejecuta un programa llamado **linker** indicándole cual es nuestro código objeto *(nombre_programa.obj).*
    - El linker genera un fichero *.exe* que ya se puede ejecutar.
    - Puede que el programa no realice correctamente la tarea que nosotros esperábamos, esto querrá decir que nos hemos equivocado en el diseño del algoritmo y deberemos revisar el paso 1.

• Compilación.

- Preprocesador de C
  - Procesa el programa fuente antes de que éste sea tratado por el compilador
  - Acciones posibles del preprocesador:
    - **Inclusión** de otros archivos en el archivo que se va a compilar
    - Definición de **constantes simbólicas**
    - Definición de macros
    - Compilación condicional de código de programa
    - Ejecución condicional de directivas de preprocesador
  - Todas las órdenes dirigidas al preprocesador sellaman **directivas** y comienzan con el símbolo #
  - Cada directiva debe estar en una línea propia y antes de cada directiva sólo pueden aparecer espacios en blanco

• Compilación.

- Definición de constantes simbólicas

#define *identificador texto_a_reemplazar*

  - *Identificador*
    - Constante que queremos definir (macro)
  - *Texto_a_reemplazar*
    - Secuencia de caracteres que va a ser representada por la macro
    - Puede incluir expresiones con operadores
  - Todas las apariciones de *identificador* en el código fuente del programa son automáticamente sustituidas por *texto_a_reemplazar* antes de la compilación del programa
    - Ejemplos:

```
#define PI 3.1416
#define DOS 2
#define CUATRO DOS*DOS
#define ERROR "Este es un mensaje de error"
```


1. Compilación.

- Inclusión de ficheros

#include <*fichero*>

#include "*fichero*"

  - En el lugar de la directiva #include se incluye una copia del archivo especificado antes de la compilación del programa
  - "*fichero*"
    - Si el nombre del archivo va entre comillas el preprocesador lo buscará en el directorio donde se encuentra el archivo que se está compilando
    - Esta forma se utiliza para incluir archivos definidos por el programador
  - <*fichero*>
    - Si el nombre del archivo va entre corchetes angulares el preprocesador lo buscará en algún directorio estándar del sistema
    - Esta forma se utiliza para incluir ficheros de cabecera de bibliotecas estándar (*stdio.h*, *math.h*,...)
    - Ejemplos:

```
#include <stdio.h>
#include "mi_fichero.h"
#include "mi_fichero.c"
```


2. Un primer programa en C. Estructura general de un programa

- Un primer programa en C: imprimir una línea de texto

```
#include <stdio.h>
main () { /*Un programa sencillo*/
        int num;
        num = 1;
        printf ("Mi número es el %d", num);
}
```

  - 1ª línea: #include <stdio.h>
    - Comunica al preprocesador que incluya dentro del programa el contenido del fichero *stdio.h*
    - Este fichero contiene información relacionada con la entrada/salida de datos (mostrar por pantalla un resultado, leer de teclado los datos que escribe un usuario, etc...)
  - 2ª línea: main()
    - Un programa en C está formado por una o más funciones
    - main() es una función que debe aparecer en cualquier programa en C
    - Todos los programas en C empiezan a ejecutarse en la función main()

2. Un primer programa en C. Estructura general de un programa

- Un primer programa en C: imprimir una línea de texto

```
#include <stdio.h>
main () { /*Un programa sencillo*/
        int num;
        num = 1;
        printf ("Mi número es el %d", num);
}
```

  - 2ª línea: /*Un programa sencillo*/
    - Es un comentario. Los programadores usan comentarios para hacer más comprensibles los programas
    - Cualquier cadena de caracteres entre /* y */ es ignorada por el compilador y no realiza ninguna acción
  - {}
    - Las llaves en este programa indican el inicio y el final de la función *main*
    - Las llaves y el trozo de programa que va entre ellas forman un **bloque**
    - Dentro de cada bloque pueden existir bloques anidados

2. Un primer programa en C. Estructura general de un programa

- Un primer programa en C: imprimir una línea de texto

```
#include <stdio.h>
main () { /*Un programa sencillo*/
        int num;
        num = 1;
        printf ("Mi número es el %d", num);
}
```

  - 3ª línea: int num;
    - Es una declaración de variables.
    - *num* es el nombre de la variable.
    - *int* indica que la variable es de tipo entero, es decir, que podrán almacenar valores que sean números enteros
    - Gracias a esta declaración, el compilador reserva en la memoria principal del ordenador espacio suficiente para almacenar la variable *num*, es decir, espacio para un número entero
  - 4ª línea: num = 1;
    - Es una sentencia de asignación. Con esta línea le damos a la variable *num* el valor *1*
    - El valor de *num* se puede modificar con otra sentencia de asignación

2. Un primer programa en C. Estructura general de un programa

- Un primer programa en C: imprimir una línea de texto

```
#include <stdio.h>
main () { /*Un programa sencillo*/
        int num;
        num = 1;
        printf ("Mi número es el %d", num);
}
```

  - 5ª línea: printf ("Mi número es el %d", num);
    - Es una llamada a una función
    - Esta línea imprime por pantalla la frase: *Mi número es el 1*
    - Indica al compilador que debe realizar la acción que corresponde a la función *printf*
    - Todo lo que aparece entre los paréntesis son los *argumentos* de la función. Los argumentos representan información que la función necesita para poder llevar a cabo su acción.
    - *%* indica que en ese lugar se va a imprimir el valor de una variable
    - *d* indica que el valor que se va a imprimir es un número entero decimal (en base 10)

3. Definición de variables y constantes. Tipos de datos fundamentales en C y sus modificadores

- Constante
  - Valor fijado antes de la ejecución del programa y que no puede ser modificado a lo largo de la misma.
  - A las constantes les pueden ser asignados nombre simbólicos mediante la directiva del preprocesador *#define:*
  - El compilador asigna automáticamente un tipo a las constantes:
    - Un número entero sin punto decimal será interpretado como *int*

      #define MAX **1000**

      X = X + **35**
    - Un número real será interpretado como *double*

      #define MAX **5.0**

      X = X – **6.2e2**
    - Un carácter entre comillas simples será usado como *char*

      #define CAR **'a'**

- Variable
  - Posición de memoria con nombre que se usa para almacenar un valor que puede ser modificado por el programa
  - Todas las variables han de estar declaradas antes de poder ser utilizadas
  - Las variables se crean y se destruyen cada vez que se entra o sale del bloque (código entre { y }) en el que son declaradas. Su contenido se pierde al salir del bloque
  - Al declarar una variable su valor es indefinido

tipo lista_de variables;

Declaración de variables en C

  - *Tipo*
    - Será un tipo de datos válido en C con algún modificador
  - *Lista_de_variables*
    - Lista de identificadores (nombres de variables) separados por comas
  - Ejemplos de declaraciones:

```
int x;
int i, j, k;
double temperatura, presion;
short int apellido,nombre;
```


- Tipos de datos
  - Cada dato de un programa pertenece a una clase llamada tipo
  - Cada tipo viene definido por:
    - Rango de valores: conjunto de valores que pueden ser almacenados en una variable declarada de ese tipo
    - Operaciones que se pueden realizar con los elementos de ese conjunto de valores
  - Tipos de datos básicos en C:
    - *char*
      - Caracteres
    - *int*
      - Números enteros
    - *float*
    - *double*
      - Números reales
    - *void*
      - Tipo nulo

- El tipo *int*
  - Representa a los números enteros con signo
  - Rango de valores:
    - Depende del ordenador y del número de bits que utilice éste para representar un entero. Normalmente se usan 16 bits por lo que el rango de valores es:

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

  - Los números enteros escritos sin punto decimal y sin exponente son reconocidos por el compilador como constantes enteras
  - Se pueden expresar números enteros en notación:
    - Decimal: 1423 5
    - Octal, empezando por 0: 02617 05
    - Hexadecimal, empezando por 0x: 0x58F 0x5

- El tipo *float*
  - Representan números reales
  - Se utiliza cuando se necesitan realizar cálculos matemáticos que requieren determinada precisión
  - Rango de valores:
    - Dependerá del método utilizado para representar los números reales en coma flotante
    - Normalmente se usan 32 bits para su representación en punto flotante, de los que se utilizan 8 para el exponente y el signo y 24 para la mantisa.
    - Este sistema permite una precisión de siete cifras decimales y un exponente cuyo rango es:

$$[10^{-37}, 10^{+37}] = [1E\text{-}37,\ 1E\text{+}37]$$

  - Los valores de tipo *float* se escriben:
    - Una serie de cifras con signo incluyendo el punto decimal + *e* ó *E* + un exponente con signo que indica la potencia de 10 a utilizar.
    - Ejemplos: -12.34e3, 0.5E-5, 1.25

- El tipo *double*
  - Representa números reales con el doble de precisión que el tipo float
  - Rango de valores:
    - Dependerá del método utilizado para representar los números reales en coma flotante
    - Normalmente se 64 bits para su representación en punto flotante. Los 32 bits de diferencia con un *float* pueden ser usados por los sistemas en:
      - La parte no exponencial, aumentando la precisión.
      - La parte exponencial, aumentando el rango de valores que pueden ser aceptados como *double*.
  - La forma de escribir valores del tipo *double* es igual que en *float*. Los números reales escritos en este formato son reconocidos por el compilador como como constantes de tipo *double*.

- El tipo *char*
  - Se utiliza para almacenar valores definidos en el juego de caracteres ASCII
  - Rango de valores:
    - El compilador de C trata a los valores de tipo *char* como si fueran un *int* de 8 bits, es decir, como un número del intervalo [-128, 127] o del [0, 255] dependiendo del compilador.
  - La asignación de valores a variables de tipo *char* puede hacerse:
    - asignándole el carácter directamente (entre comillas simples):
      char letra = 'A';
    - asignándole el código ASCII del carácter:
      char letra = 65;
  - Es importante distinguir entre un carácter numérico y un número:

```
char caracter = 4;  /* carácter ASCII número 4, que se corresponde
                       con el carácter ' ♦' */
char caracter = '4'  /* caracter '4', que se corresponde con el
                        carácter ASCII número 52 */
```


- El tipo *char*
  - Existen caracteres especiales (algunos de ellos no son imprimibles) que se representan por un carácter antecedido por \. Los más usuales son:

Carácter	ASCII	Acción
\n	10	nueva línea
\t	9	tabulador
\b	8	retroceso
\r	13	retorno de carro
\f	12	salto de página
\\	92	barra atrás
\'	39	apóstrofo
\"	34	comillas
\a	7	alerta
\0	0	Carácter nulo. Fin de cadena de caracteres.

  - Estos caracteres se asignan igual que los demás:

```
char caracter;          char caracter;
caracter = '\t'         caracter = 9;
```


- El tipo *void*
  - Tipo *nulo* o *vacio*
  - Se usa para:
    - Declarar funciones que no devuelven ningún valor
    - Crear punteros genéricos

  su uso se estudiará en temas posteriores.

3. Definición de variables y constantes. Tipos de datos fundamentales en C y sus modificadores

- Modificadores de tipos
  - Se usan para alterar el significado del tipo base de modo que se ajuste más a las necesidades del programa
  - Modificadores de tipos en C:
    - *unsigned*
      - utiliza el mismo número de bits del tipo al que modifica para representar sólo valores positivos.
        Por ejemplo,
        int con 16 bits tiene el rango [-32.768, +32.767]
        unsigned int tiene el rango [0, 65.535]
    - *signed*
      - utiliza el mismo número de bits del tipo al que modifica para representar valores positivos y negativos
        Por ejemplo,
        char con 8 bits tiene el rango [-127, 127]
        signed char tiene el rango [0, 255]
    - *short*
      - disminuye el número de bits utilizado para representar el valor y por lo tanto su rango.
    - *long*
      - aumenta el número de bits utilizado para representar el valor y por lo tanto su rango.
  - No siempre *short* y *long* disminuyen o aumentan el número de bits para representar el valor, esto depende del compilador. El lenguaje C siempre nos garantiza:

    short <= tipo <= long

3. Definición de variables y constantes. Tipos de datos fundamentales en C y sus modificadores

Combinaciones de tipos que se ajustan al estándar de C

TIPO	Tamaño aproximado en bits	Intervalo mínimo
char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
int	16 o 32	-32.768 a 32.767
unsigned int	16 o 32	0 a 65.535
signed int	16 o 32	Igual que int
short int	16	-32.768 a 32.767
unsigned short int	16	0 a 65.535
signed short int	16	Igual que short int
long int	32	-2.147.483.647 a 2.147.483.647
signed long int	32	Igual que long int
unsigned long int	32	0 a 4.294.967.295
float	32	1E-37 a 1E+37 con seis dígitos de precisión
double	64	1E-37 a 1E+37 con diez dígitos de precisión
long double	80	1E-37 a 1E+37 con diez dígitos de precisión

4. Operadores

- Operador de asignación **=**

  nombre_de_variable = expresión;
  - Asigna a la variable a la izquierda del = el valor resultante de evaluar la expresión de la derecha.

    resultado = (x + y – z) / 4;

    mi_var = 5;
  - **Asignaciones múltiples**
    - Se puede asignar a muchas variables a la vez el mismo valor

      nombre_de_variable1 = nombre_de_variable2 = ... = expresión;

      resultado = entrada = salida = (x+y-z) / 4

      x = y = z = 0;
  - **Asignaciones compuestas**
    - Se le asigna a la variable el valor que se obtiene como resultado de operar *(op)* su antiguo valor con el valor de la expresión a su derecha
    - *op* puede ser cualquier operador aritmético

      nombre_de_variable1 ***op*= expresión;**

      x += 10 equivale a x = x + 10

      x -= 5 equivale a x = x –5

      x *= 3 * y + 45 equivale a x = x * (3 * y + 45)

4. Operadores

- Operadores aritméticos

Operador	Acción
- (binario y monario)	Resta
+ (binario y monario)	Suma
*	Multiplicación
/	División
%	Módulo (resto de una división entera)
++	Preincremento Postincremento
--	Predecremento Postdecremento

4. Operadores

- Operadores aritméticos
  - Incremento **++**, Decremento **--**
    - Aumenta o disminuye el valor de su operando en una unidad
    - Puede colocarse como prefijo (pre...) o como sufijo (post...).
    - **Pre** incremento, **Pre** decremento
      - 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.
    - **Post** incremento , **Post** decremento
      - 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).

Operación	Resultado
++x	**x = x + 1**
--x	**x = x – 1**
x = 100; y = ++x;	**El valor de y es 101**
x = 100; y = x++;	**El valor de y es 100**
cont = 4; ++cont < 5;	**Falso**
cont = 4; cont++ < 5;	**Verdadero**

Ejemplos de uso de los operadores **++** y **--**

4. Operadores

- Operadores relacionales
  - Se emplean para hacer comparaciones
  - Se usan para construir las expresiones empleadas en las sentencias de selección e iteración
  - El resultado de una expresión con operadores relacionales es siempre **0** (falso) o **1** (cierto)
  - En C:
    - **Cierto** es todo aquel valor **distinto de 0**
    - **Falso** es el valor **0**

      a = 58;

      if (a != 0)... devuelve 1 (cierto)

      if (a)... también devuelve 1 (cierto)
  - Para las comparaciones de **números reales** es mejor **no usar** operadores con **igualdad**, o lo que es lo mismo, usar sólo **>** y **<**, ya que pueden no ser iguales debido a errores de redondeo aunque en realidad deberían serlo.

Operador	Acción
<	Menor estricto
<=	Menor o igual
==	Igual
!=	Distinto
>=	Mayor o igual
>	Mayor

4. Operadores

- Operadores lógicos
  - Se emplean para combinar dos o más expresiones relacionales
  - Se usan para construir las expresiones empleadas en las sentencias de selección e iteración

Operador	Acción
&&	Y
||	O
!	NO

  - En C las expresiones lógicas se evalúan de izquierda a derecha. En el momento en que un elemento invalide la expresión completa cesa la evaluación de la misma.

```
num = 0;
(num != 0) && (num/3 == 1)
```

    - Se evalúa a 0 (FALSO)
    - No se llega a evaluar
    - FALSO && ...cualquier cosa es FALSO...
    - Luego no hace falta seguir evaluando la expresión

4. Operadores

Precedencia de los operadores de C

De mayor a menor...
( ) [ ] ®
! ~ ++ -- - (tipo) * & sizeof
*** / %**
+ -
< <= > >=
== !=
&
^
&&
||
?:
= += *= /=

4. Operadores

- Conversión de tipos
  - Cuando en una expresión se mezclan constantes y variables de distintos tipos el compilador convierte de forma **automática** todo a un único tipo siguiendo las siguientes reglas:
    - **Promoción**: en cualquier operación en la que aparezcan dos tipos diferentes se eleva el rango del que lo tiene menor para igualarlo al del mayor.
    - El rango o categoría de los tipos de mayor a menor es el siguiente:
      - double, float long, int, short, char
      - Los tipos *unsigned* tienen el mismo rango que los tipos a los que están referidos
    - En una sentencia de asignación, el resultado final de los cálculos se reconvierte al tipo de la variable al que está siendo asignado. 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.

4. Operadores

- Conversión de tipos
  - **Casting**
    - Es una conversión **explícita** del tipo de una expresión realizada por el programador
    - La forma general de realizarla es: **(tipo)expresión;**

```
int x;
float y = 1.0;
float z = 2.0;
x = (int) y + (int) z
```


4. Operadores

Resultados de las conversiones de tipos más usuales

Tipo destino	Tipo de expresión	Posible pérdida de información
signed char	char	Si valor > 127, destino negativo
char	short int	8 bits más significativos
char	int (16 bits)	8 bits más significativos
char	int (32 bits)	24 bits más significativos
char	long int	24 bits más significativos
short int	int (16 bits)	Nada
short int	int (32 bits)	16 bits más significativos
int (16 bits)	long int	16 bits más significativos
int (32 bits)	long int	Nada
int	float	Parte fraccional y posiblemente más
float	double	Precisión, resultado redondeado
double	long double	Precisión, resultado redondeado

5. Operaciones de Entrada/Salida (stdio.h)

5.1. Salida de datos: la función printf()

- Salida de datos
  - Tanto la entrada como la salida de datos en C se realizan a través de funciones de la librería *stdio.h*
  - La salida de datos con formato se realiza con la función *printf()* cuyo prototipoes:

int printf (const char *formato, argumento,...);

  - *Formato*
    - Es una cadena que contiene
      - Los caracteres que se mostrarán en pantalla
      - Especificadores de formato:
        - %código-formato
        - Definen la forma en que se muestran los argumentos posteriores
        - Tendrá que haber tantos especificadores de formato como argumentos aparezcan en la llamada a la función
  - *Argumentos*
    - Expresiones que aparecerán en la salida en el lugar de los especificadores de formato

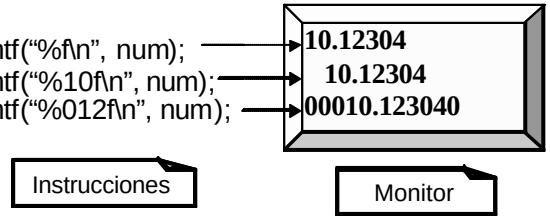
Especificadores de formato de printf más usuales

Código	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 decimal sin signo (letras mayúsculas)
%o	Entero	Entero octal sin signo
%f	Real	[-] dddd.dddd
%e	Real	[-] d.ddde[+|-]ddd
%E	Real	[-] d.dddE[+|-]ddd
%g	Real	Usa %e o %f, el más corto
%G	Real	Usa %E o %f, el más corto
%c	Carácter	Un solo carácter
%s	Puntero a cadena	Cadena de caracteres

- Modificadores de formato
  - Alteran ligeramente el significado de los especificadores de formato
  - **Longitud mínima de campo**
    - Es un entero situado entre el signo % y el código de formato que hace que se rellene la salida con espacios a partir del borde izquierdo de la pantalla
    - Si lleva un 0 delante se rellena con 0's en vez de con espacios
    - Si la cadena o el número es más largo que ese mínimo se mostrará completamente

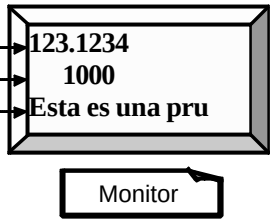
```
double num;
num = 10.12304;

printf("%f\n", num);
printf("%10f\n", num);
printf("%012f\n", num);
```



- Modificadores de formato
  - **Especificador de precisión**
    - Sigue al especificador de longitud mínima de campo (si existe)
    - Consiste de un punto seguido de un entero
    - Su significado exacto depende del tipo al que se aplique
      - Con %f, %e, %E indica el número de posiciones decimales a mostrar
      - Con %g, %G indica el número de dígitos significativos
      - Con %s indica el máximo de caracteres que se mostrarán

```
printf("%.4f\n", 123.1234567);
printf("%3.8df\n", 1000);
printf("%10.15s\n", "Esta es una prueba.");
```



- Modificadores de formato
  - **Ajuste de la salida**
    - Cuando se utilizan en un formato longitud mínima de campo y/o precisión, todos los valores se ajustan a la derecha (se rellena con espacios por la izquierda).
    - El signo – después de % indica que el ajuste se realice a la izquierda

5. Operaciones de Entrada/Salida (stdio.h)

5.2. Entrada de datos: la función scanf()

- Entrada de datos
  - La entrada de datos con formato a través del teclado se realiza con la función *scanf()* cuyo prototipo es:

```
int scanf ( const char *formato,&variable,...);
```

  - *Formato*
    - Es una cadena formada por especificadores de formato (%código-formato) que determinan el tipo de la entrada esperada
  - *Variable*
    - Sólo llevan & las variables que son de tipos básicos
    - Si la variable es una cadena de caracteres no lleva &
  - La función *scanf* va almacenando en las variables lo que el usuario va introduciendo por teclado
  - *scanf* considera que dos ítem de entrada son diferentes cuando están separados por:
    - Blancos, Tabulados, Caracteres de nueva línea, Espacios en blanco
- Los especificadores de formato son los mismos que para *printf* excepto:
  - No existe la opción %g
  - Las opciones %f y %e son equivalentes
  - Existe una opción %h para leer enteros short

5. Operaciones de Entrada/Salida (stdio.h)

5.3. Otras funciones de E/S

- Entrada / salida de un solo carácter
  - Lee un solo carácter de la entrada estándar (*stdin)*
  - Devuelve un valor que es el carácter leído y que puede ser asignado a una variable de tipo *char* o *int*
  - Guarda la entrada hasta que se pulsa *ENTER*
  - Se encuentra en ***stdio.h***

```
int getchar (void);
```

  - Lee un solo carácter del teclado
  - No hace eco en la pantalla
  - Se encuentra en ***conio.h***

```
int getch (void);
```

  - Lee un solo carácter del teclado
  - Hace eco en la pantalla
  - Se encuentra en ***conio.h***

```
int getche (void);
```

  - Lee una cadena de caracteres de *stdin*
  - Sustituye el carácter '\n' por el carácter '\0' de fin de cadena
  - Permite que las cadenas tengan espacios en blanco y tabuladores
  - Se encuentra en ***stdio.h***

```
char *gets (char *s);
```

  - Escribe un carácter por pantalla
  - Se encuentra en ***stdio.h***

```
int putchar (char c);
```


5. Operaciones de Entrada/Salida (stdio.h)

5.4. EJEMPLOS

```
int main () {
   int c;
   while ((c=getchar() != '\n') {
        putchar(c);
   }
}
```

```
int main () {
   char c;
   c=getche();
   printf("Ha tecleado el carácter '%c'",c);
}
```

```
int main () {
   int edad;
   float sueldo;
   printf("Escriba su edad y sueldo.\n");
   scanf("%d %f", &edad, &sueldo);
   printf("\nEdad : %d Sueldo: %.2f \n", edad, sueldo);
}
```


5. Funciones matemáticas (math.h)

- `int abs (int);` • Devuelve el **valor absoluto** de un entero
- `double cos (double);` • Devuelve el **coseno**
- `double sin (double);` • Devuelve el **seno**
- `double exp (double x);` • Devuelve la exponencial: e^x
- `double fabs (double);` • Devuelve el **valor absoluto** de un real
- `double pow (double x, double y);` • Devuelve la potencia de x elevado a y: x^y
- `double sqrt (double x);` • Devuelve la **raiz cuadrada** positiva de x. Si x es negativo devuelve 0
- `double log (double x);` • Devuelve el **logaritmo neperiano**
- `double log10 (double x);` • Devuelve el **logaritmo en base 10**

Bibliografía

- Bibliografíaprincipal
  - J.M. Rodriguez , J. Galindo. "Aprendiendo C". (2a. Edición). Servicio de Publicaciones de la Universidad de Cádiz, 1997.
  - James L. Antonakos. "Programación estructurada en C". Prentice-Hall, 2000
  - Herbert Schildt. "C: guía de autoenseñanza". McGraw Hill, 1999
- Bibliografía adicional
  - Kernighan, Brian W. Dennis M.Ritchie. "El lenguaje de programación C". (2a. Edición). Prentice-Hall, 1991
  - P.J. Sánchez S., J. Galindo, I. Turias D., I. Lloret G. "Ejercicios Resueltos de Programación C". Servicio de Publicaciones de la Universidad de Cádiz, 1998.
  - Schildt, Herbert. "C: Manual de referencia". (2a. Edición). McGraw Hill, 1993.