

## PROGRAMACIÓN

### (GRADOS EN INGENIERO MECÁNICO, ELÉCTRICO, ELECTRÓNICO INDUSTRIAL y QUÍMICO INDUSTRIAL)

<b>Sesión</b>	6 (Diseño modular. Sintaxis de C: estructura modular de un programa)	
<b>Temporización</b>	1 hora (no presencial)	
<b>Objetivos formativos</b>	<ul style="list-style-type: none"> <li>• Conocer la técnica de diseño modular para estructurar las soluciones a problemas complejos.</li> <li>• Conocer y entender la notación sub-algorítmica utilizada para la definición de los módulos. Conocer y diferenciar entre los diferentes tipos de módulos en base a su actividad funcional (entrada, salida, control, transformación,...) y los resultados que devuelven (procedimiento y función).</li> <li>• Saber diferenciar entre la definición de un módulo y su activación. Distinguir entre parámetros formales y parámetros reales (o actuales).</li> <li>• Entender y saber utilizar los diferentes mecanismos de comunicación de información entre módulos.</li> <li>• Distinguir entre parámetros de E, S y E/S. Entender las diferencias entre paso de parámetros por valor (o copia) y por dirección (puntero o variable).</li> <li>• Conocer la estructura sintáctica de un programa modular en C: definiciones de funciones, prototipos de funciones, bibliotecas de funciones, paso de parámetros, tipo de dato puntero, funciones como parámetros, tipos de almacenamiento automático, externo, estático y registro.</li> </ul>	
<b>Competencias a desarrollar</b>	<ul style="list-style-type: none"> <li>• RD1: Poseer y comprender conocimientos</li> <li>• RD2: Aplicación de conocimientos</li> <li>• UAL1: Conocimientos básicos de la profesión</li> <li>• UAL3: Capacidad para resolver problemas</li> <li>• UAL6: Trabajo en equipo</li> <li>• FB3: Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en la ingeniería.</li> </ul>	X X X X  X
<b>Materiales</b>	Sesiones de teoría 6.1 a 6.6 + bibliografía tema 3 + Internet	
<b>Tarea</b>	Estudiar los conceptos de diseño modular y la sintaxis de C para implementarlos. Realizar los ejercicios propuestos.	
<b>Fecha de entrega</b>	<b>Inicio sesión 6 de Grupo de Trabajo.</b> Compare los resultados de los ejercicios realizados con sus compañeros de equipo de trabajo y discuta con ellos posibles discrepancias. Si surgen dudas consulte con el profesor.	
<b>Criterios de éxito</b>	<ul style="list-style-type: none"> <li>• Terminar en el tiempo previsto la tarea.</li> <li>• Demostrar, en una prueba escrita u oral, mediante las respuestas a las preguntas del profesor que ha alcanzado los objetivos formativos.</li> </ul>	
<b>Plan de trabajo</b>	<b>Actividad</b>	<b>Temporización</b>
	Estudio de los contenidos de las sesiones de teoría 6.1 a 6.6, sobre los conceptos de diseño modular y la sintaxis de C para implementarlos.	20 mn
	Lectura detallada de las páginas 120 a 134 del libro de N.	10 mn

	Dale y C. Weems "Pascal - 2ª edición", ed. McGraw-Hill, 1989. Nota: puede encontrar este libro en la biblioteca de la Universidad.	
	Realizar los ejercicios propuestos y elaboración de la documentación a presentar según modelo adjunto.	30 mn

# Sintaxis de C: Diseño Modular: estructura modular de un programa

## 1.- Estructura modular de un programa en C.

```

/* Bibliotecas utilizadas: operaciones y cálculos de uso común */
#include <stdio.h>
#include <math.h>
#include ...

/* Prototipos de funciones */
...

/* Definiciones de funciones */
main ...
función1...
función2...
...
funciónN...

```

Diagram illustrating the modular structure of a C program:

- The first three lines (comments and `#include` directives) are grouped by a bracket and labeled "Bibliotecas de funciones externas".
- The next two lines (comment and `...`) are grouped by a bracket and labeled "Funciones definidas por el usuario".
- The last line (`funciónN...`) is labeled "Función principal".

Un programa C consta de un conjunto de definiciones de funciones, que incluye como mínimo una función llamada **main** que es la que inicia la ejecución del programa. Cada una de las funciones contiene una serie de instrucciones y realiza una tarea específica. Dentro de una función pueden encontrarse instrucciones para llamar (activar o ejecutar) a otras funciones. Una función puede llamarse a sí misma (recursividad).

Como la funcionalidad directa del lenguaje C es muy reducida, ciertas operaciones, como son las entradas y salidas, se incorporan mediante funciones externas, que se suministran mediante bibliotecas ajenas al compilador. Éste es el caso de las operaciones de entrada y salida que se encuentran en una biblioteca estándar de nombre **stdio.h**. La directiva **#include**, le indica al entorno de trabajo que ciertas operaciones se encuentran en las bibliotecas de funciones.

## - Definición de una función.

La estructura de una función es la siguiente:

```

tipo_resultado nombre_función(declaración de parámetros formales){
    declaraciones e instrucciones si existen;
    return expresión; /* valor devuelto por la función */
}
/* a través de su nombre de identificador */

```

**tipo\_resultado:** tipo de dato del valor devuelto por la función a través de su identificador o nombre. Puede ser un tipo simple (**int**, **char**, **float**), un puntero a cualquier tipo o una estructura (**struct**).

**nombre\_función:** identificador de la función, utilizando las mismas reglas que para los otros identificadores.

**Declaración de parámetros formales (o argumentos):** declaración de los elementos que se transfieren a la función desde la parte del programa que hace la llamada:  
`tipo1 arg1, tipo2 arg2, ..., tipoN argN`  
 Los identificadores utilizados para los argumentos son locales en el sentido de que no son reconocidos fuera de la función.

**return expresión;** Devuelve el valor de la expresión (resultado de la función) al punto de llamada. Devuelve el control al punto de llamada.

Las funciones siempre devuelven un valor a través de su identificador (nombre de la función) que puede ser útil ó no. Si la función devuelve un valor no entero, en su definición debe ir precedido su nombre por el tipo que devuelve; en el caso de que la función no devuelva ningún resultado a través de su identificador (procedimiento) es conveniente preceder su nombre con la palabra reservada **void** e incluir una instrucción **return;** al final de la misma.

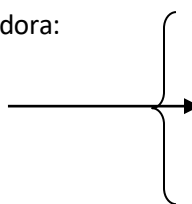
El acceso a una función se realiza especificando su nombre seguido de una lista de argumentos encerrados entre paréntesis y separados por comas (o un par de paréntesis vacíos si la función no tiene ningún argumento):

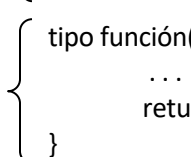
`nombre_función(lista de parámetros reales o actuales)`

La llamada a una función puede formar parte de una instrucción simple o puede ser uno de los operandos de una expresión. La definición de una función debe preceder a la llamada a la misma, o alternativamente debe declararse previamente a la llamada un prototipo de función (es usual colocar los prototipos de todas las funciones que se definen al principio del programa):

Prototipo de función llamada: `tipo función(declaración de parámetros formales);`

Definición función llamadora:

Llamada a función: 

Definición de función llamada: 

Nótese que el prototipo de función es equivalente a la primera línea de la definición de la función añadiéndole un ";" al final.

<b>Estructura del programa: correspondencia entre pseudo-código y sintaxis de C</b>		
	<b>Pseudo-código</b>	<b>Sintaxis de C</b>
<b>Módulo principal</b> - Constantes globales  - Tipos de datos globales - Variables globales  - Instrucciones del algoritmo	ALGORITMO nombre algoritmo	
	CONST    n=5 Max=100 C=3.45	#define n     5 #define Max 100 #define C    3.45
	TIPOS nombreTipo: definiciónTipo	typedef definiciónTipo nombreTipo;
	VAR    var1, var2:tipo1 var3, var4: tipo2	tipo1 var1, var2; tipo2 var3, var4; /* externas: declaradas fuera de las funciones. También se pueden declarar en la función main(), pero en este caso solo serían accesibles dentro de esta función */
	INICIO    instrucción 1 instrucción 2 ... instrucción N FIN_ALGORITMO_PRINCIPAL	int main(){ instrucción 1; ... instrucción N; return 0; }
<b>Procedimiento</b> -Variables locales  - Instrucciones	PROCEDIMIENTO nombre_p (lista par. For)	void nombre_p(lista par. Formales){
	VAR    var1, var2: tipo1	tipo1 var1, var2;
	INICIO    instrucción 1 instrucción 2 ... instrucción N FIN_PROCEDIMIENTO	instrucción 1; instrucción 2; ... instrucción N; }
<b>Función</b> -Variables locales  - Instrucciones	FUNCION nombre_f (lista par. Form): tipo_res	tipo_res nombre_f(lista par. Form.){
	VAR    var1, var2: tipo1	tipo1 var1, var2;
	INICIO    instrucción 1 instrucción 2 ... instrucción N Devolver (expresión)	instrucción 1; instrucción 2; ... instrucción N; return (expresión);
	FIN_FUNCION	}

## 2.- Comunicación entre funciones.

La comunicación de información entre funciones se puede realizar a través de:

- El valor especial devuelto por la función a través del identificador (nombre) de la función:  
    return (expresión) ;
- Los argumentos de la función (lista de parámetros formales).
- Variables externas (no se recomienda su uso en programación modular por los posibles efectos laterales, esto es la propagación de errores que pudiera surgir).

### - Tipo de dato puntero.

Un puntero es una variable que contiene la dirección de otra variable. Declaración de una variable puntero:

tipo \*var;

\*var es una variable de ese tipo (var es un puntero a variables de ese tipo).

Operador & devuelve la dirección de un objeto (variable, elemento de un vector,...). No es aplicable a expresiones ó **register**.

Operador \* toma su operando como una dirección y accede a la dirección en cuestión, dándonos su contenido.

Los punteros como argumentos de funciones suelen emplearse para el caso de las funciones que devuelven valores a través de su lista de argumentos (paso de parámetros por variable, dirección, salida o entrada/salida).

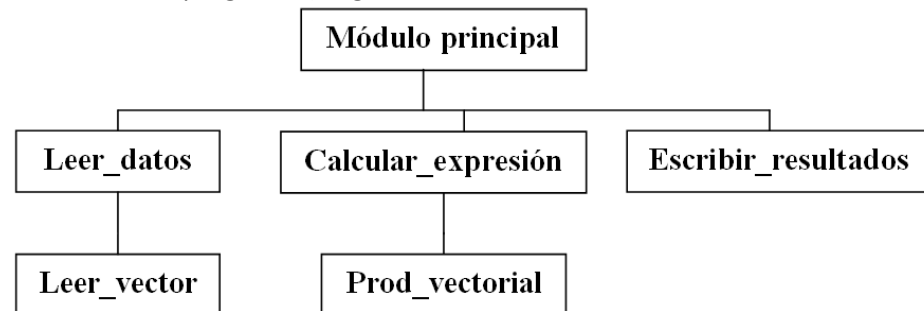
### - Paso de parámetros.

Comunicación entre funciones a través de la lista de parámetros formales o argumentos: en la llamada se establece una correspondencia automática entre los parámetros actuales y los parámetros formales → deben de coincidir en cantidad, tipo (tipo de dato y tipo de parámetro) y orden.

Tipo de parámetro	Pseudo-código	Sintaxis de C
E (valor o copia)	Par. Formal: pf: tipo (E) Par. Actual: pa pa: cte, variable o expresión pf: variable local que recibe una copia del valor del parámetro actual (cambios en la copia no afectan al valor original)	Par. Formal: tipo pf Par. Actual: pa
		Excepciones: no está permitido el paso de arrays (o cadenas de caracteres) por valor. En algunos compiladores tampoco se permite el paso de estructuras y uniones por valor.
S o E/S (variable o dirección)	Par. Formal: pf: tipo (S ó E/S) Par. Actual: pa pa: variable pf: representa el contenido del parámetro actual; cambios en el mismo afectan al parámetro actual	Par. Formal: tipo *pf Par. Actual: &pa pa: variable &pa: dirección de memoria de la variable actual pf: puntero al parámetro actual *pf: contenido del parámetro actual
		Arrays y cadenas de caracteres: Par. Formal: tipo_vector pf Par. Actual: pa pa: el nombre del array representa la dirección del primer elemento pf: array que representa el contenido del parámetro actual.

### 3.- Ejemplo de programa con diseño modular.

<b>Ejercicio</b>	Construir un programa que calcule e imprima en pantalla el resultado de la siguiente expresión, donde los componentes de los vectores A, B y C se introducen por teclado: $D = A \wedge (B \wedge C)$ ( $\wedge$ : producto vectorial)
------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Diseño**Diseño preliminarDiseño de datosEstructura del programa (diagrama de módulos)Interfaces entre módulos

Nombre módulo	Tipo de parámetro	Nombre parámetro	Tipo dato
Módulo principal			
Leer_datos	S	ax,ay,az	real
	S	bx,by,bz	real
	S	cx,cy,cz	real
Leer_vector	S	vx,vy,vz	real
Calcular_expresión	E	ax,ay,az	real
	E	bx,by,bz	real
	E	cx,cy,cz	real
	S	dx,dy,dz	real
Prod_vectorial	E	ax,ay,az	real
	E	bx,by,bz	real
	S	cx,cy,cz	real
Escribir_resultados	E	vx,vy,vz	real

Diseño detallado

## Algoritmo módulo\_principal

```
{ programa que calcula la expresión:  $D = A \wedge (B \wedge C)$  }
```

```
Var ax,ay,az: real
```

```
bx,by,bz: real
```

```
cx,cy,cz: real
```

```
dx,dy,dz: real
```

```
Inicio Llamar a leer_datos(ax,ay,az,bx,by,bz,cx,cy,cz)
```

```
Llamar a calcular_expresion(ax,ay,az,bx,by,bz,cx,cy,cz,dx,dy,dz)
```

```
Llamar a escribir_resultados(dx,dy,dz)
```

```
Fin_algoritmo_principal
```

```
Procedimiento leer_datos(ax:real(S), ay:real(S), az:real(S), bx:real(S),  
by:real(S), bz:real(S), cx:real(S), cy:real(S), cz:real(S))
```

```
Inicio Escribir("Introduzca vector A:");
```

```
Llamar a leer_vector(ax,ay,az)
```

```
Escribir("Introduzca vector B:");
```

```
Llamar a leer_vector(bx,by,bz)
```

```
Escribir("Introduzca vector C:");
```

```
Llamar a leer_vector(cx,cy,cz)
```

```
Fin_procedimiento
```

	<pre> Procedimiento leer_vector(vx:real(S), vy:real(S), vz:real(S)) Inicio  Escribir("x: ")         Leer(vx)         Escribir("y: ")         Leer(vy)         Escribir("z: ")         Leer( vz) Fin_procedimiento  Procedimiento calcular_expresion(ax:real(E), ay:real(E), az:real(E),                                 bx:real(E), by:real(E), bz:real(E), cx:real(E), cy:real(E), cz:real(E),                                 dx:real(S),dy:real(S),dz:real(S)) Var     ex,ey,ez: real Inicio  Llamar a prod_vectorial(bx,by,bz,cx,cy,cz,ex,ey,ez)         Llamar a prod_vectorial(ax,ay,az,ex,ey,ez,dx,dy,dz) Fin_procedimiento  Procedimiento prod_vectorial(ax:real(E), ay:real(E), az:real(E),                              bx:real(E), by:real(E), bz:real(E), cx:real(S), cy:real(S), cz:real(S)) Inicio  cx←ay*bz-az*by         cy←az*bx-ax*bz         cz←ax*by-ay*bx Fin_procedimiento  Procedimiento escribir_resultados(vx:real(E), vy:real(E), vz:real(E)) Inicio  Escribir("Componentes del vector D:")         Escribir("x= ",vx)         Escribir("y= ",vy)         Escribir("z= ",vz) Fin_procedimiento </pre>
<b>Codificación</b>	<pre> /* Programa que calcula: D=A^(B^C)          */ #include &lt;stdio.h&gt; #include &lt;stdlib.h&gt; #include &lt;conio.h&gt; #include &lt;ctype.h&gt; #include &lt;math.h&gt;  /* Prototipos de funciones */ void leer_datos(float *ax, float *ay, float *az,                float *bx, float *by, float *bz,                float *cx, float *cy, float *cz); void leer_vector(float *vx, float *vy, float *vz); void calcular_expresion(float ax, float ay, float az,                        float bx, float by, float bz,                        float cx, float cy, float cz,                        float *dx, float *dy, float *dz); void prod_vectorial(float ax, float ay, float az,                    float bx, float by, float bz,                    float *cx, float *cy, float *cz); void escribir_resultados(float vx, float vy, float vz);  /* Definiciones de funciones */ int main(){     char c;     float ax,ay,az;     float bx,by,bz;     float cx,cy,cz;     float dx,dy,dz;      do{ system("cls"); </pre>



	<pre> printf("CALCULO EXPRESION D=A^(B^C)\n"); printf("=====\n\n"); leer_datos(&amp;ax,&amp;ay,&amp;az,&amp;bx,&amp;by,&amp;bz,&amp;cx,&amp;cy,&amp;cz); calcular_expresion(ax,ay,az,bx,by,bz,cx,cy,cz,&amp;dx,&amp;dy,&amp;dz); escribir_resultados(dx,dy,dz); printf("\n\nDesea efectuar una nueva operacion (s/n)? "); c=toupper(getch()); }while (c!='N'); return 0; }  void leer_datos(float *ax, float *ay, float *az, float *bx, float *by, float *bz, float *cx, float *cy, float *cz){ printf("\nIntroduzca vector A:\n"); leer_vector(ax,ay,az); printf("\nIntroduzca vector B:\n"); leer_vector(bx,by,bz); printf("\nIntroduzca vector C:\n"); leer_vector(cx,cy,cz); }  void leer_vector(float *vx, float *vy, float *vz){ printf("\tx: "); scanf("%f", vx); printf("\ty: "); scanf("%f", vy); printf("\tz: "); scanf("%f", vz); }  void calcular_expresion(float ax, float ay, float az, float bx, float by, float bz, float cx, float cy, float cz, float *dx, float *dy, float *dz){ float ex,ey,ez;  prod_vectorial(bx,by,bz,cx,cy,cz,&amp;ex,&amp;ey,&amp;ez); prod_vectorial(ax,ay,az,ex,ey,ez,dx,dy,dz); }  void prod_vectorial(float ax, float ay, float az, float bx, float by, float bz, float *cx, float *cy, float *cz){ *cx=ay*bz-az*by; *cy=az*bx-ax*bz; *cz=ax*by-ay*bx; }  void escribir_resultados(float vx, float vy, float vz){ printf("\nComponentes del vector D:\n"); printf("\tx= %.1f\n",vx); printf("\ty= %.1f\n",vy); printf("\tz= %.1f\n",vz); } </pre>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### 4.- Bibliotecas de funciones.

Funciones de E/S		#include <stdio.h>
printf(...)	int	Escritura en pantalla con formato de salida.
scanf(...)	int	Lectura de datos por teclado. Devuelve el número de conceptos leídos ( <b>EOF</b> si falla).
getchar()	int	Lectura de 1 carácter del teclado con buffer. Ejemplo:       char c; c=getchar(); La marca fin de archivo <b>EOF</b> se corresponde con el código -1, por lo que resulta más conveniente que <b>c</b> sea de tipo <b>int</b> .

putchar(c)	int	Imprime un carácter (argumento). Ejemplo: <code>putchar(getchar());</code>
gets(s)	char*	Lee una cadena de caracteres del teclado, admite espacios en blanco y finaliza cuando se pulsa <b>INTRO</b> . Coloca la marca <b>\0</b> al final de la cadena. Devuelve un valor ( <b>0</b> ó <b>NULL</b> si hay problemas). Ejemplo: <code>char cadena[20]; gets(cadena);</code>
puts(s)	int	Escribe en pantalla una cadena (hasta <b>\0</b> ). Después de escribir salta de línea. Ejemplo: <code>puts(cadena);</code>
sprintf(s,...)	int	imprime salida formateada a una cadena (en lugar del monitor). Sintaxis: <code>sprintf(cadena,"cadena_control",argumentos);</code>
sscanf(s,...)	int	Lectura de entrada formateada de una cadena (en lugar del teclado). Sintaxis: <code>sscanf(cadena,"cadena_control",argumentos);</code>
<b>Funciones de E/S #include &lt;conio.h&gt; (Biblioteca no estándar)</b>		
clrscr()	void	Borra la pantalla (ventana de texto de salida) y posiciona el cursor a la esquina superior izquierda (compilador de Borland C++).
getch()	int	Lectura de 1 carácter del teclado sin buffer. Ejemplo: <code>c=getch();</code>
<b>Funciones que manejan caracteres #include &lt;ctype.h&gt;</b>		
toascii(c)	int	Convierte el valor del argumento a ASCII.
toupper(c)	int	Convierte un carácter a mayúsculas. Ejemplo: <code>sal=toupper(ent);</code>
tolower(c)	int	Convierte un carácter a minúsculas.
isalpha(c)	int	Devuelve un valor distinto de 0 (cumple la condición) si el argumento es un carácter alfabético. Ejemplo: <code>i=isalpha(car);</code>
isdigit(c)	int	Devuelve un valor distinto de 0 si el argumento es un dígito.
islower(c)	int	Devuelve un valor distinto de 0 si el argumento es una letra minúscula.
isspace(c)	int	Devuelve un valor distinto de 0 si el argumento es un espacio en blanco.
isupper(c)	int	Devuelve un valor distinto de 0 si el argumento es una letra mayúscula.
isalnum(c)	int	Devuelve un valor distinto de 0 si el argumento es un carácter alfanumérico.
isascii(c)	int	Devuelve un valor distinto de 0 si el argumento es un carácter ASCII (0-127).
isctrl(c)	int	Devuelve un valor distinto de 0 si el argumento es un carácter de control.
ispunct(c)	int	Devuelve un valor distinto de 0 si el argumento es un signo de puntuación.
isgraph(c)	int	Devuelve un valor distinto de 0 si el argumento es un carácter ASCII gráfico (hex 0x21-0x7e; octal 041-176).
isodigit(c)	int	Devuelve un valor distinto de 0 si el argumento es un dígito octal.
isxdigit(c)	int	Devuelve un valor distinto de 0 si el argumento es un dígito hexadecimal.

Funciones matemáticas		#include <math.h>
acos(d)	double	Arco coseno (0-pi) del argumento (-1,+1).
asin(d)	double	Arco seno (-pi/2,+pi/2) del argumento (-1,+1).
atan(d)	double	Arco tangente (-pi/2,+pi/2) del argumento.
atan2(d1,d2)	double	Devuelve el arco tangente de d1/d2.
ceil(d)	double	Devuelve el entero más pequeño mayor ó igual al argumento (redondeo hacia arriba).
cos(d)	double	Coseno del argumento expresado en radianes.
cosh(d)	double	Coseno hiperbólico del argumento.
exp(d)	double	Exponencial del argumento.
fabs(d)	double	Valor absoluto de un número real.
floor(d)	double	Redondeo hacia abajo (mayor entero menor ó igual a d).
fmod(d1,d2)	double	Devuelve el resto de d1/d2 con el mismo signo que d1.
labs(l)	long int	Valor absoluto de un entero largo.
log(d)	double	Logaritmo natural del argumento.
log10(d)	double	Logaritmo decimal del argumento.
pow(d1,d2)	double	Calcula d1 elevado a d2.
sin(d)	double	Seno del argumento expresado en radianes.
sinh(d)	double	Seno hiperbólico del argumento.
sqrt(d)	double	Raíz cuadrada positiva del argumento.
tan(d)	double	Tangente del argumento expresado en radianes.
tanh(d)	double	Tangente hiperbólica del argumento.
Funciones varias		#include <stdlib.h>
abs(i)	int	Valor absoluto de un entero.
system(s)	int	Pasa la orden s al sistema operativo. Devuelve 0 si la orden se ejecuta con éxito. Ej: <code>system("cls");</code>

Nota: la segunda columna indica el tipo de dato devuelto por la función. En la primera columna, los argumentos que aparecen son:

c	un carácter
d	doble precisión
f	archivo
i	entero
l	entero largo
p	puntero
s	cadena
u	entero sin signo

<b>Asignatura</b>	Programación		
<b>Plan de Estudios</b>	Grados en Ingeniero Mecánico, Eléctrico, Electrónico Industrial y Químico Industrial		
<b>Actividad</b>	Trabajo individual	<b>Sesión</b>	6
<b>Tiempo empleado</b>			

<b>Apellidos, nombre</b>	<b>DNI</b>	<b>Firma</b>

### 1.- Ejercicios.

1. Vaya a la biblioteca de la Universidad de Almería y en la sala de lectura solicite el texto de N. Dale y C. Weems "Pascal - 2ª edición", ed. McGraw-Hill, 1989. Lea detenidamente las páginas 120 a 134, y a continuación resuma con sus propias palabras qué entiende por diseño descendente.
2. ¿Es obligatorio utilizar módulos para construir un programa que resuelva un determinado problema?
3. ¿De cuantas funciones puede constar un programa escrito en lenguaje C? ¿Es posible iniciar la ejecución del programa con una función cualquiera del programa?
4. Relacione los siguientes elementos de la documentación del diseño con los correspondientes de un programa escrito en lenguaje de programación C (inclusión de bibliotecas externas, definición de constantes, prototipos de funciones, definiciones de funciones, almacenamiento externo, automático,...).

Documentación del diseño		Programa en C
Diseño preliminar	Diseño de datos	typedef → se estudiará más adelante
	Estructura del programa	
	Interfaces entre módulos	
Diseño detallado	Sub-algoritmos	
Módulo principal		
Variable global (accesible desde cualquier módulo)		
Variable local		

5. Los módulos representan secuencias de instrucciones con una funcionalidad limitada y específica. ¿Cuáles son las principales actividades funcionales que realiza un módulo? ¿Es posible que un módulo realice varias tareas funcionales?
6. ¿Qué diferencias encuentra entre una función y un procedimiento? ¿Existen los procedimientos en lenguaje de programación C?
7. ¿Qué significa activar o llamar a un módulo, desde el punto de vista del control del procesador central (flujo de ejecución)?
8. ¿Qué mecanismos existen para transferir información entre módulos?
9. ¿Qué relación encuentra entre los parámetros formales y los parámetros reales cuando se activa un módulo?
10. ¿Para qué se utilizan los parámetros de E, de S y/o de E/S que aparecen en la interfaz de un módulo (1ª línea de la definición del módulo)? ¿Qué son los parámetros de E? ¿Y los parámetros de S?
11. Supongamos que tenemos definido un módulo con la siguiente interfaz:

*Procedimiento proc\_exa(x: entero (E), y: entero (E), c: carácter (E), z: real (S))*

Indicar cuáles de las siguientes llamadas (activaciones del módulo) son correctas y cuáles no, justificando brevemente la respuesta:

```

Var    x,y: entero
        c: carácter
        z: real
    . . .
    x ← 4
    y ← 5
    c ← 'C'
1) z ← proc_exa(x+y,2,'D')
2) proc_exa(x,c,y,z)
3) proc_exa(y,x,c,z)
4) proc_exa(2*x,y,c,2*z)

```

12. Supongamos que tenemos definido un módulo con la siguiente interfaz:

*Función func\_exa(x: entero(E), y: entero(E), c: carácter (S), z: real(S)):entero*

Indicar cuáles de las siguientes llamadas (activaciones del módulo) son correctas y cuáles no, justificando brevemente la respuesta:

```

Var    x,y: entero
        c: carácter
        z: real
    . . .
    x ← 4
    y ← 5
    c ← 'C'
1) func_exa(y,x,c,z)
2) Escribir(func_exa(x+y,x-y,'D',z))
3) x←func_exa(x,x,c,z)
4) x←func_exa(x,func_exa(x,5,c,z),c,z)

```

13. Para cada uno de los siguientes módulos, indicar los resultados (o los errores) que se producen en las activaciones (llamadas) correspondientes:

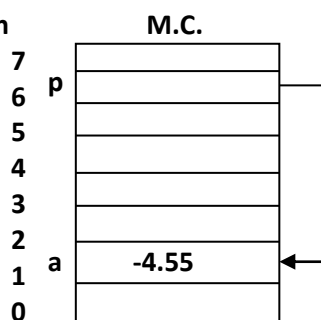
Definición del módulo	Activación
Procedimiento Saludo() Inicio Escribir("Hola, soy un módulo") Escribir("sin parámetros") Fin_procedimiento	Var x: entero ... x←3 Saludo Saludo(x) Saludo() Escribir(Saludo())
Función Suma(n:entero(E)):entero Var i,suma: entero Inicio i←1 suma←0 Mientras(i≤n) Hacer suma←suma+i i←i+1 Fin_mientras Devolver(suma) Fin_función	Var x,y: entero ... x←4 y←5 ... Escribir(Suma(3)) Escribir(Suma(y)+Suma(x)) Escribir(Suma(x+y)) Escribir(Suma(Suma(2)))

<p>Función F(a:entero(E),c:carácter(E)):entero</p> <p>Var x:entero</p> <p>Inicio Según_sea (c) Hacer</p> <p>  '+': <math>x \leftarrow a+a</math></p> <p>  '-': <math>x \leftarrow -a</math></p> <p>  Sino: <math>x \leftarrow a</math></p> <p>  Fin_según_sea</p> <p>  Devolver(x)</p> <p>Fin_función</p>	<p>Var x: entero</p> <p>s: caracter</p> <p>...</p> <p><math>x \leftarrow 4</math></p> <p><math>s \leftarrow '+'</math></p> <p>...</p> <p>Escribir(F(x+2, '+'))</p> <p>Escribir(F(F(x, '-'), s))</p> <p>Escribir(F(x, '+')*F(-3, '-')/F(1, 'v'))</p> <p>Escribir(F('+',5))</p>
<p>Procedimiento Intercambia(x:real(E),y:real(E))</p> <p>Var aux: real</p> <p>Inicio <math>aux \leftarrow x</math></p> <p>      <math>x \leftarrow y</math></p> <p>      <math>y \leftarrow aux</math></p> <p>Fin_procedimiento</p>	<p>Var a,b: real</p> <p>...</p> <p><math>a \leftarrow 3</math></p> <p><math>b \leftarrow 5</math></p> <p>Escribir("Valores originales: ")</p> <p>Escribir("a=",a, "b=",b)</p> <p>Intercambia(a,b)</p> <p>Escribir("V. intercambiados: ")</p> <p>Escribir("a=",a, "b=",b)</p>
<p>Procedimiento Normalizar(vx:real(E),vy:real(E))</p> <p>Var v:real</p> <p>Inicio <math>v \leftarrow \text{raiz2}(vx*vx+vy*vy)</math></p> <p>      <math>vx \leftarrow vx/v</math></p> <p>      <math>vy \leftarrow vy/v</math></p> <p>Fin_procedimiento</p>	<p>Var vx: real</p> <p>vy: real</p> <p>Inicio ....</p> <p><math>vx \leftarrow 8</math></p> <p><math>vy \leftarrow 6</math></p> <p>Llamar a Normalizar(vx,vy)</p> <p>Escribir("V. Normalizado:")</p> <p>Escribir("(" ,vx, " ,",vy, ")")</p>
<p>Procedimiento Swap(x:real(E/S),y:real(E/S))</p> <p>Var aux: real</p> <p>Inicio <math>aux \leftarrow x</math></p> <p>      <math>x \leftarrow y</math></p> <p>      <math>y \leftarrow aux</math></p> <p>Fin_procedimiento</p>	<p>Var a,b: real</p> <p>...</p> <p><math>a \leftarrow 3</math></p> <p><math>b \leftarrow 5</math></p> <p>Escribir("Valores originales: ")</p> <p>Escribir("a=",a, "b=",b)</p> <p>Swap(a,b)</p> <p>Escribir("V. intercambiados: ")</p> <p>Escribir("a=",a, "b=",b)</p>
<p>Procedimiento Swap(x:real(E/S),y:real(E/S))</p> <p>Var aux: real</p> <p>Inicio <math>aux \leftarrow x</math></p> <p>      <math>x \leftarrow y</math></p> <p>      <math>y \leftarrow aux</math></p> <p>Fin_procedimiento</p>	<p>Var x,y: real</p> <p>Inicio ....</p> <p><math>x \leftarrow 4</math></p> <p><math>y \leftarrow 5</math></p> <p>...</p> <p>Llamar a Swap(y,x)</p> <p>Escribir(x)</p> <p>Llamar a Swap(x,3)</p> <p>Escribir(x)</p>

14. Supongamos que tenemos definido en un programa C una variable **a** de tipo real (simple precisión) y otra **p** de tipo puntero a real. Indicar el valor de las expresiones indicadas, suponiendo que se han ejecutado previamente las dos siguientes instrucciones:

```
float a, *p;
a=-4.55;
p=&a;
```

Dirección



Expresión	Valor expresión
a	
p	
*p	
&a	
&*p	
&p	
*a	

**2.- Resultados de aprendizaje (reflexión):** marque con una cruz los objetivos que cree haber alcanzado tras realizar esta actividad, y rellene en el campo de observaciones aquellos aspectos que cree que necesita mejorar (si los hubiera):

Objetivos formativos		Cumplimiento
<ul style="list-style-type: none"> <li>Conocer la técnica de diseño modular para estructurar las soluciones a problemas complejos.</li> <li>Conocer y entender la notación sub-algortmica utilizada para la definición de los módulos. Conocer y diferenciar entre los diferentes tipos de módulos en base a su actividad funcional (entrada, salida, control, transformación,...) y los resultados que devuelven (procedimiento y función).</li> <li>Saber diferenciar entre la definición de un módulo y su activación. Distinguir entre parámetros formales y parámetros reales (o actuales).</li> <li>Entender y saber utilizar los diferentes mecanismos de comunicación de información entre módulos.</li> <li>Distinguir entre parámetros de E, S y E/S. Entender las diferencias entre paso de parámetros por valor (o copia) y por dirección (puntero o variable).</li> <li>Conocer la estructura sintáctica de un programa modular en C: definiciones de funciones, prototipos de funciones, bibliotecas de funciones, paso de parámetros, tipo de dato puntero, funciones como parámetros, tipos de almacenamiento automático, externo, estático y registro.</li> </ul>		
Observaciones		

## Respuestas ejercicios

- 1) iiii Leer las páginas indicadas !!!
- 2) No es obligatorio.
- 3) De todas las que considere necesario.  
No (los programas en C siempre se inician con la función **main()**).
- 4)

Documentación del diseño		Programa en C
Diseño preliminar	Diseño de datos	typedef → se estudiará más adelante
	Estructura del programa	
	Interfaces entre módulos	Prototipos de funciones
Diseño detallado	Sub-algoritmos	Funciones
Módulo principal		Función main()
Variable global (accesible desde cualquier módulo)		Variable externa (declarada fuera de cualquier función)
Variable local		Variable automática (declarada dentro de la definición de una función)

- 5) Actividades funcionales básicas: entrada, cálculo (transformación), salida, control.  
Si es posible.
- 6) La principal diferencia es que las funciones devuelven un valor especial a través de su identificador y los procedimientos no. Otra diferencia está en la activación: la llamada a una función debe aparecer como operando dentro de una expresión, mientras que la llamada a un procedimiento es una instrucción terminada en ;  
En C los procedimientos son funciones **void** (nulo: no devuelve ningún valor a través de su identificador).
- 7) El control del procesador central se transfiere a la primera instrucción del módulo.
- 8) Tres mecanismos:
  - Valor especial devuelto a través del identificador de la función.
  - Variables globales.
  - Lista de parámetros formales (argumentos del módulo).
- 9) Deben coincidir en cantidad, orden y tipo (tipo de dato y tipo de parámetro).
- 10) Para transferir información entre módulos (el llamador y el llamado).  
E: el módulo llamado recibe elementos de información del módulo llamador.  
S: el módulo llamado devuelve resultados al módulo llamador.  
E/S: el módulo llamado recibe elementos de información que puede devolver modificados.

11)

1)	<code>z ← proc_exa(x+y, 2, 'D')</code>	Incorrecta: es un procedimiento y no una función, y falta el último parámetro real.
2)	<code>proc_exa(x, c, y, z)</code>	Incorrecta: el segundo parámetro real debe de ser entero y el tercero de tipo carácter.
3)	<code>proc_exa(y, x, c, z)</code>	Correcta.
4)	<code>proc_exa(2*x, y, c, 2*z)</code>	Incorrecta: el cuarto parámetro real debe de ser una variable y no una expresión.



12)

1)	<code>func_exa(y,x,c,z)</code>	Incorrecta: la llamada a una función debe de aparecer como un operando dentro de una expresión.
2)	<code>Escribir(func_exa(x+y,x-y,'D',z))</code>	Incorrecta: el tercer parámetro real debe de ser una variable y no una expresión.
3)	<code>x←func_exa(x,x,c,z)</code>	Correcta.
4)	<code>x←func_exa(x,func_exa(x,5,c,z),c,z)</code>	Correcta

13)

Activaciones	Resultados en pantalla
Saludo	Error de sintaxis: faltan los paréntesis
Saludo(x)	Error de sintaxis: el módulo no tiene argumentos
Saludo()	Hola, soy un módulo sin parámetros
Escribir(Saludo())	Error de sintaxis: la llamada a un procedimiento no es una expresión (es una instrucción)
Escribir(Suma(3))	6
Escribir(Suma(y)+Suma(x))	25
Escribir(Suma(x+y))	45
Escribir(Suma(Suma(2)))	6
Escribir(F(x+2, '+'))	12
Escribir(F(F(x, '-'), s))	-8
Escribir(F(x, '+')*F(-3, '-')/F(1, 'v'))	24
Escribir(F('+',5))	Error de sintaxis: no coinciden posicionalmente los tipos de datos de los parámetros reales con los de los parámetros formales
Escribir("Valores originales: ") Escribir("a=",a, "b=",b) Intercambia(a,b) Escribir("V. intercambiados: ") Escribir("a=",a, "b=",b)	Valores originales: a=3 b=5  V. intercambiados: a=3 b=5
Llamar a Normalizar(vx,vy) Escribir("V. Normalizado:") Escribir("(" ,vx, " ,",vy, ")")	V. Normalizado: (8.0,6.0)
Escribir("Valores originales: ") Escribir("a=",a, "b=",b) Swap(a,b) Escribir("V. intercambiados: ") Escribir("a=",a, "b=",b)	Valores originales: a=3 b=5  V. intercambiados: a=5 b=3

Llamar a Swap(y,x) Escribir(x) Llamar a Swap(x,3)	5 Error de sintaxis: el segundo parámetro real debe de ser una variable
---------------------------------------------------------	----------------------------------------------------------------------------

14)

Dirección

M.C.

7		
6	p	
5		
4		
3		
2		
1	a	-4.55
0		

Expresión	Valor expresión
a	-4.55
p	1
*p	-4.55
&a	1
&*p	1
&p	6
*a	Error de sintaxis