# Ejercicios de programación en C:
# Relación de Ejercicios 1

Programación de Sistemas para Control de Procesos.
Ingeniería Técnica en Electrónica Industrial.

1. Efectuar un programa que lea los valores de 3 resistencias electrónicas (en Ohmios, Ω) conectadas en paralelo y muestre en pantalla el valor global de las 3. El valor global es calculado por la siguiente ecuación: $1 / (1/R_1 + 1/R_2 + 1/R_3)$.

2. Modificar el programa anterior para aplicar la fórmula a cualquier cantidad de resistencias. Primeramente, el programa pedirá el número de resistencias a conectar en paralelo. Después pedirá uno a uno los valores de todas las resistencias. Por último, escribirá el resultado global.

3. Modifique el programa anterior para que muestre también el resultado global si todas las resistencias fueran conectadas en serie (en vez de en paralelo). Recuerde que si conectamos en serie n resistencias el resultado global es la suma de todas ellas.

4. Efectuar un programa que lea dos números enteros y muestre en pantalla todos los números comprendidos entre dichos números, ambos incluidos. El programa no debe suponer que el primero será menor que el segundo, ni viceversa, pero sí que deberá tenerlo en cuenta para mostrar los números en orden creciente o decreciente según corresponda.

   Modificar el programa anterior para que los números sean mostrados por filas, de forma que cada fila tenga n números. El número de números por filas, n, será leído al principio del programa. Cada número puede ir separado del siguiente por el carácter tabulador (ASCII número 9).

5. Implementar lo siguiente:

   - Una función que, dados (como argumentos por Valor) los valores de 2 resistencias (en ohmios), devuelva la resistencia global que ofrecen ambas si se conectan en paralelo.

   - Un programa que calcule la resistencia global de n resistencias conectadas en paralelo. El programa pedirá sucesivamente una tras otra las n resistencias. El programa entenderá que no hay más resistencias cuando lea una resistencia menor o igual a cero. El programa se implementará de forma que utilice la función anterior.

6. Escribir una función que, dados (como argumentos por Valor) los valores de 3 resistencias (en ohmios), devuelva la resistencia global que ofrecen si se conectan en paralelo. Implementar esta función usando y sin usar la función del ejercicio anterior.

7. Implementar una función, `Digit(N,num)` que devuelva el dígito `N`-ésimo de un número `num` de tipo `unsigned long int`, teniendo en cuenta que el dígito 0 es el dígito más a la derecha (el menos significativo). La función devolverá -1 si el número no tiene suficientes dígitos. Considere la posibilidad de que `N` pueda ser un número negativo. Ejemplos:

| | |
|---|---|
| `Digit (0,3456)` | Devuelve 6 |
| `Digit (1,-3456)` | Devuelve 5 |
| `Digit (4,3456)` | Devuelve -1 |

8. Las resistencias electrónicas suelen ir identificadas por un código de colores que permite marcar cada resistencia con su valor (en Ohmios, Ω) y su Tolerancia (en %). Este código de colores viene representado en la siguiente tabla:

| *Dígito* | *Color* | *Multiplicador* | *Tolerancia* |
|---|---|---|---|
| | Ninguno | | 20% |
| | Plata | 0.01 | 10% |
| | Oro | 0.1 | 5% |
| **0** | Negro | 1 | |
| **1** | Marrón | 10 | |
| **2** | Rojo | $10^2$ | 2% |
| **3** | Naranja | $10^3$ | |
| **4** | Amarillo | $10^4$ | |
| **5** | Verde | $10^5$ | |
| **6** | Azul | $10^6$ | |
| **7** | Violeta | $10^7$ | |
| **8** | Gris | | |
| **9** | Blanco | | |

El código que suele emplearse en las resistencias es un código de 4 colores, es decir, cada resistencia está marcada con 4 bandas y cada una de ellas puede ser de diferente color. Cada banda tiene un significado, que depende de cada color:

- Las primeras 2 bandas indican un número de 2 dígitos: Esos dos dígitos vienen dados por el color de esas bandas, según la columna "Dígito" de la tabla.
- La tercera banda es un valor por el que se multiplicará el número obtenido por las bandas anteriores. Una vez multiplicados ambos valores, obtenemos el valor de la resistencia en Ohmios (Ω).
- La cuarta banda indica la tolerancia de la resistencia y, como puede verse en la tabla, no puede ser de cualquier color.

Ejemplo: Unas resistencias con los siguientes colores, tienen los siguientes valores de resistencia y tolerancia:

Verde-Azul-Amarillo-Oro 560kΩ, 5%
Rojo-Negro-Rojo-Rojo 2kΩ, 2%
Rojo-Rojo-Marrón-Plata 220Ω, 10%

Según todo lo anterior, implemente un subprograma que permita calcular la resistencia y la tolerancia de una resistencia, sabiendo los códigos de colores. El subprograma tendrá, como mínimo, 4 argumentos, que serán números naturales, y que indicarán el color de las bandas según la columna "Dígito". Los colores Oro, Plata y Ninguno tomarán los valores 10, 11 y 12 respectivamente.

Implementar otro subprograma que muestre por pantalla el dígito que le corresponde a cada color (incluyendo los dígitos 10, 11 y 12).

Implementar también un programa que pida los colores de las 4 bandas y muestre los valores devueltos por el anterior subprograma. El programa mostrará el dígito que le corresponde a cada color usando el procedimiento ya creado y leerá de teclado 4 números que corresponderán a los colores de las 4 bandas.

Tras esta lectura mostrará los datos de la resistencia con esos colores en las bandas. El programa se repetirá indefinidamente hasta que lea un valor negativo como color de una banda.

9. Hacer un programa que dibuje una hélice en una posición determinada de la pantalla. El programa pedirá primero los valores X e Y de esa posición, indicando un error si alguno de esos dos valores está fuera del rango de la pantalla. Después, borrará la pantalla (con la función `clrscr()` de la librería `conio.h`) y situará el cursor en la posición introducida (con la función `gotoxy()` de la librería `conio.h`), escribiendo sucesivamente los siguientes 4 caracteres, de forma que al llegar al último escriba de nuevo el primero:

```
'-'  '\'  '|'  '/'
```

Al poner en una posición fija de la pantalla esos 4 caracteres sucesiva y repetidamente da la sensación de que es una hélice girando. La velocidad de la hélice puede controlarse introduciendo una espera entre cada dos estados de la hélice (por ejemplo con la función `Delay()` de la librería `dos.h`). La duración de esa espera puede ser también leída del teclado, al principio del programa, igual que la posición. El programa terminará cuando se pulse la tecla de escape ESC (carácter 27 del código ASCII).

10. Modificar el programa anterior para que la posición de la hélice no sea fija, sino que esté establecida a un valor inicial y podamos modificar esa posición utilizando las flechas del teclado, de forma que, por ejemplo, si se pulsa la flecha DERECHA sume una unidad en la posición X, desplazando así la hélice hacia la derecha.

NOTA 1: Controlar que no se exceda de los límites de la pantalla.
NOTA 2: Al mover la hélice, recuerde que tiene que borrar el carácter que hubiera en la posición anterior.
NOTA 3: Para averiguar cuales son los códigos de las teclas de flechas puede hacer un programa que lea caracteres de teclado y escriba sus respectivos códigos. Con este programa ejecutándose, pulse las flechas y observe cuales son sus códigos. Tenga en cuenta que las flechas (como las teclas de función F1, F2...) están formadas por 2 caracteres, el primero es el carácter nulo (ASCII 0) y el segundo indica la tecla pulsada.

11. Modificar el programa anterior para incorporar que se puede modificar la velocidad de la hélice durante la ejecución del programa. Para ello, se incorporará una espera en cada paso (por ejemplo con la función `Delay()` de la librería `dos.h`), de forma que el argumento de esta función varíe sumando o restando un incremento fijo INC (definido como una constante simbólica), de la siguiente forma:

- Si se pulsa '+' se incrementará el argumento en INC.
- Si se pulsa '*' se incrementará el argumento en INC+INC.
- Si se pulsa '-' se decrementará el argumento en INC.
- Si se pulsa '/' se decrementará el argumento en INC+INC.

NOTA: El argumento de la función de espera, puede declararse como un `unsigned char`, de forma que si se incrementa demasiado retorne la cuenta a cero y nunca se haga la espera excesivamente larga.

12. Utilizar una declaración de tipos para una **matriz** de números reales de tamaño DIM×DIM, donde DIM es una constante, declarada como tal, de valor 10. Escribir las siguientes funciones:

a) **Función `traspuestaM`**: Acepta una matriz como único argumento y devuelve su matriz traspuesta.
b) **Función `simetricaM`**: Acepta una matriz como único argumento y devuelve 1 si dicha matriz es simétrica y 0 si no lo es. Para averiguar si la matriz es o no simétrica se debe usar la función **`traspuestaM`** del apartado anterior.

c) **Función `sumaM`**: Acepta tres matrices devolviendo en la tercera de ellas la suma de las dos primeras.
d) **Función `restaM`**: Acepta tres matrices devolviendo en la tercera de ellas la resta de las dos primeras.
e) **Función `multiplicaM`**: Acepta tres matrices devolviendo en la tercera de ellas la multiplicación de las dos primeras.

13. Hacer un programa que muestre por pantalla la **representación en binario** de distintos tipos de datos: int, long int, unsigned int y char. El programa pedirá un valor del tipo en cuestión y mostrará la representación de dicho valor en binario. Para ello se deben usar los operadores a nivel de bits, accediendo a cada bit individualmente y mostrando su valor (0 ó 1).

14. Hacer una función sin argumentos que cada vez que se llame devuelva el siguiente valor de la sucesión de **Fibonacci**. O sea, la primera vez que se llame devolverá 0, la siguiente 1 y a partir de la tercera llamada devolverá la suma de los dos valores anteriores: 1, 2, 3, 5, 8, 13, 21, 34... La función devolverá datos de tipo `unsigned long int`. ¿Hasta qué término es posible calcular sin que se produzca desbordamiento?. Intente modificar la función para que posibilite llegar a un término mayor utilizando datos de tipo `long double`.

15. **Calendario Perpetuo:** El ejercicio 16 del tema 5 tiene por objetivo averiguar el día de la semana del 1 de Enero de cualquier año posterior a 1582. Basándose en ese ejercicio hacer un programa que muestre en pantalla el calendario de un mes y año elegido por el usuario del programa. El programa mostrará el calendario del mes elegido de forma tradicional, es decir, en 7 columnas, una para cada día de la semana y teniendo en cuenta el número de días de cada mes, prestando especial cuidado con el mes de Febrero por si es año bisiesto.

    El programa deberá remarcar de alguna forma los días festivos: Todos Domingos y algunas fiestas especiales 28 de Febrero (día de Andalucía), 1 de Mayo (día del trabajo), 12 de Octubre (día de la Hispanidad, Virgen del Pilar), 6 y 8 de Diciembre (día de la Constitución y de la Inmaculada), 25 de Diciembre (Navidad)... y también el Jueves y el Viernes Santo. La fecha de estos últimos dos días se calculará según indica el ejercicio 20 del tema 4.

    La mejor forma de remarcar los días festivos es utilizando la función `setcolor()` para cambiar el color de la biblioteca `graphics.h`. Por ejemplo, para establecer el color a rojo debe usarse `setcolor(RED)`. La mejor forma de aprender a usar esta función es buscar esta función en la ayuda del compilador y examinar y probar el ejemplo que incluye. También puede probarse a cambiar el tipo y tamaño de letra con la función `settextstyle()`.

16. **Máximo Multiplicador Cabalístico:** Hay números naturales que al multiplicarlos sucesivamente por 1, 2, 3, 4... se obtienen números que tienen los mismos dígitos que el original pero en distinto orden (sólo al multiplicar por 1 se obtiene un número con los mismos dígitos en el mismo orden). Supongamos que para el número N, se cumple esa propiedad al multiplicarlo por 1, 2, 3..., X. Es decir, si yo multiplico N por cualquier número del intervalo [1,X], el resultado será un número con los mismos dígitos que N, pero en distinto orden. Entonces, decimos que el "**máximo multiplicador cabalístico**" de N es X: **MMC(N)=X**.

    Ejemplo: **MMC(142857) = 6;** → Por ejemplo, 142857*5=714285. Para el número 142857 la propiedad se cumple, además de para el número 5, también para cualquier número del intervalo [1,6].

    Hacer un programa que muestre (usando funciones independientes):
    - La mayor lista de números posible, en la que todos cumplen que MMC(N) es mayor o igual a 2. Al final debe mostrar el N cuyo valor MMC(N) es el mayor de todos los números analizados.

- Dado un número N, mostrar su valor MMC(N).

17. **Comparación de cadenas en español:** Implementar la función `ComparaCads()` que se comportará de forma similar a la función `strcmp()` de la biblioteca `string.h`, pero para **comparación de cadenas en español**. La función tendrá tres argumentos. Los dos primeros argumentos de la función serán dos cadenas de caracteres. Una cadena es "menor" que otra cuando está antes siguiendo el orden alfabético.

    Para este ejercicio no se puede utilizar la biblioteca `string.h`. La función `ComparaCads` debe solucionar dos problemas que tiene la función `strcmp()` para comparar texto en español:

    - Debe comparar correctamente cualesquiera palabras en español: Considerando como iguales las vocales si están acentuadas (mayúsculas o minúsculas), la letra "u" con diéresis ("ü" y "Ü") y teniendo en cuenta la letra "ñ" (y "Ñ"), está situada tras la letra "n" en el abecedario español pero está situada en otra posición en la tabla ASCII.
    - La función tendrá un tercer argumento que será de tipo entero: Si su valor es 1 (`TRUE`) la comparación se efectuará teniendo en cuenta que las mayúsculas y las minúsculas son iguales, o sea, al comparar "hola" y "HOLA" devolverá 0. Si el valor de este último argumento es (`FALSE`), no considerará que las mayúsculas y las minúsculas son iguales.

18. **Dividir calculando periodos:** Implementar un programa para **dividir** dos números de tipo `long double`. El resultado debe tener hasta 10 decimales, redondeando este último según el undécimo decimal. El programa debe detectar e indicar si entre los 10 primeros decimales se produce algún **periodo** (puro o mixto). El Menú contendrá las opciones "Cambiar dividendo: X" (donde X es el último dividendo introducido), "Cambiar divisor: Y" (donde Y es el último divisor introducido) y "Calcular división". Por defecto, al principio del programa se debe suponer un valor cualquiera para el dividendo y el divisor.
    **Ejemplos:** 12/9 = 1.3 con 3 como periodo (1.3333333333...).
    12/7 = 1.714285 con 714285 como periodo (1. 714285714285714285...).
    13/6 = 2.16 con 6 como periodo (2.166666666...).

    Compruebe que el divisor no sea nunca cero. Para efectuar diversas divisiones cambiando sólo el divisor bastará con usar la segunda y tercera opciones sucesivamente, ya que el dividendo permanecerá constante mientras no se cambie usando la primera opción. Para calcular los periodos se pueden calcular los restos obtenidos al sacar decimales en la división y, en cada resto comprobar si ese resto ya se ha repetido anteriormente, en cuyo caso tenemos un periodo a partir de ese dígito.

19. Quizás el más famoso de todos los sistemas de codificación es el **código Morse**, desarrollado por Samuel Morse en 1832, para uso en el sistema telegráfico. El código Morse asigna una serie de puntos y rayas a cada letra del alfabeto, a cada dígito y a unos cuantos caracteres especiales. La separación entre palabras se indica por un espacio o por la ausencia de un punto o una raya. La versión internacional del código Morse aparece en la tabla siguiente:

| Carácter | Código | Carácter | Código |
|---|---|---|---|
| A | .- | T | - |
| B | -... | U | ..- |
| C | -.-. | V | ...- |
| D | -.. | W | .-- |
| E | . | X | -..- |
| F | ..-. | Y | -.-- |
| G | --. | Z | --.. |
| H | .... | | |
| I | .. | **Números** | |

| | | | |
|---|---|---|---|
| J | .--- | 1 | .---- |
| K | -.- | 2 | ..--- |
| L | .-.. | 3 | ...-- |
| M | -- | 4 | ....- |
| N | -. | 5 | ..... |
| O | --- | 6 | -.... |
| P | .--. | 7 | --... |
| Q | --.- | 8 | ---.. |
| R | .-. | 9 | ----. |
| S | ... | 0 | ----- |

Escriba un programa que lea una frase escrita en español y cifre dicha frase en código Morse y que también lea una frase en código Morse y la convierta en el equivalente en español. Utilice un espacio en blanco entre cada letra codificada Morse y tres espacios en blanco entre cada palabra codificada en Morse.

El programa deberá incorporar una función Menu() que muestre las siguientes opciones: 1) Pasar una frase a código Morse, lo cual se implementará en una función que se llame Frase2Morse(), 2) Pasar código Morse a una frase, implementando una función que se llame Morse2Frase() y 3) Salir.

20. **Análisis de texto.** La disponibilidad de computadoras con capacidades de manipulación de cadenas nos proporciona interesantes métodos para analizar lo escrito por grandes autores. Se ha puesto, por ejemplo, gran atención al hecho de saber si William Shakespeare alguna vez existió. Algunos estudiosos creen que existen evidencias indicando que Christopher Marlowe fue el que escribió las obras maestras atribuidas a Shakespeare. Los investigadores han utilizado ordenadores para localizar similitudes en los textos de estos dos autores. Realice un programa que lea varias líneas de texto y analice las siguientes características del texto:

a) Imprimir una tabla indicando el número de veces que aparece cada letra del alfabeto en dicho texto.

b) Imprimir una tabla que indique el número de palabras de una letra, de dos letras, de tres letras ... que aparecen en el texto.

c) Imprimir una tabla indicando el número de ocurrencias de cada palabra distinta en el texto. Para ello supondremos que el texto tiene como máximo 100 palabras distintas, con lo que deberá almacenarlas en un array de estructuras de tamaño 100. Cada estructura deberá contener una cadena de caracteres con la palabra (máximo 20 caracteres) y otro campo con el número de veces que aparece esa palabra en el texto. Considere opcionalmente la posibilidad de que las palabras aparezcan ordenadas alfabéticamente. Para ordenarlas puede utilizar cualquier algoritmo de ordenación teniendo en cuenta que hay que intercambiar estructuras completas (la cadena y el número de ocurrencias). Para comparar las cadenas de caracteres en la ordenación utilice la función `strcmp()` que dice si dos cadenas son iguales, si una es mayor que la otra o viceversa.

El programa deberá mostrar un menú con las siguientes opciones: 1) Introducir texto, 2) Número de instancias de cada letra, 3) Número de palabras de cada longitud, 4) Número de ocurrencias de cada palabra distinta, 5) Salir.

21. Escribir una función que devuelva 1 si la cadena Source del primer argumento concuerda con el valor de la cadena Pattern del segundo argumento. El patrón puede contener los caracteres comodines '*' y '?'. Un carácter '*' concuerda con cualquier secuencia de caracteres (incluyendo una secuencia de longitud 0) y un carácter '?' concuerda con cualquier carácter (sólo uno).

Construye un programa de prueba que muestre un menú con las siguientes opciones:

- Introducir cadena.
- Introducir patrón.
- Concordancia: dice si la cadena actual concuerda o no con el patrón actual.
- Salir.

Por ejemplo, supón el patrón “ab*o”. Concuerdan con él todas las cadenas que empiecen por “ab”, luego tengan cualquier número de caracteres (incluso 0) y terminen con una “o” . Concuerdan: abo, abanico, abeto, abuelo... y no concuerdan: abuela, avecilla, hola...

20. Escribir una función que efectúe la misma operación que la función `itoa(valor,cadena,base)` de la biblioteca `stdlib.h`, pero evitando dos limitaciones que tiene esa función. Por una parte, permitir el que número del primer argumento sea de tipo `long int`. Por otra parte, tras el argumento de la cadena la nueva función aceptará un nuevo argumento que indique la longitud máxima de dicha cadena, de forma que la función no podrá escribir más caracteres que los que ahí se indiquen, incluso aunque el resultado los requiera. Además, la nueva función devolverá un entero que indicará si la conversión se efectuó correctamente (1) o si la conversión no se efectuó correctamente por falta de espacio en la cadena resultante. Observe que la nueva función tendrá 4 argumentos, de modo que el último sigue siendo la base a la que se desea convertir el número entero del primer argumento.

# Ejercicios de Memoria Dinámica

**1.** Programar las funciones primitivas siguientes, para una estructura de datos dinámica **lista** CON y SIN cabecera. Supondremos que cada elemento de una lista sólo tiene un entero largo:

- Inicializar lista.
- Ver si la lista está vacía.
- Longitud de la lista.
- Insertar en la lista en una determinada posición.
- Insertar por orden.
- Borrar un elemento dando su posición en la lista.
- Borrar todos los elementos de la lista.
- Leer el elemento, dando su posición.
- Obtener la posición del primer elemento que tiene un determinado valor.
- Modificar un valor concreto, dando su posición en la lista.
- Escribir en pantalla todos los valores que contenga cada elemento de la lista (en nuestro caso es sólo un entero).
- Mostrar todos los elementos de la lista, utilizando la primitiva anterior.

Todas las primitivas enumeradas deberán ser programadas en funciones independientes. Las primitivas de inserción devolverán el valor −1 si no hubo memoria suficiente para efectuar tal inserción y 0 si la inserción se efectuó correctamente. Efectuar también un programa principal con un menú que incorpore todas las primitivas enumeradas anteriormente. Para cada opción, lógicamente pedirá los valores que se requieran y tendrá que tener en cuenta si la operación se efectuó con éxito o no. Antes de terminar el programa deberá liberarse toda la memoria ocupada por la lista.

**2.** Programar, utilizando memoria dinámica, las primitivas de un **lista doblemente enlazada sin cabecera y circular** de forma similar al ejercicio anterior. O sea, cada elemento tendrá un puntero al siguiente elemento y otro al elemento anterior, de manera que el elemento siguiente al último es el primero y el elemento anterior al primero es el último. La lista incluirá un dato que marcará el **elemento actual**, que será un puntero a uno de los elementos de la lista. Esto es, una lista de este tipo consiste en dos punteros: Uno al primer elemento y otro al elemento actual. Además de las primitivas básicas expuestas en el ejercicio anterior se dispondrán de primitivas para las siguientes tareas

- Borrar y Modificar el elemento actual de la lista.
- Leer el elemento actual: Recuperar su valor.
- Ir al siguiente elemento y al anterior: Esto cambiará el elemento actual por el siguiente/anterior.
- Insertar un elemento en la posición actual: Esto desplazará todos los elementos una posición, a partir del elemento actual.
- Intercambiar dos elementos: Esto intercambiará de posición el elemento actual con el siguiente, de forma que la posición del elemento actual no se modifica, pero si se modifican lógicamente sus valores.
- Ordenar la lista: Utilizando exclusivamente las primitivas anteriores implemente algún algoritmo para ordenar la lista (burbuja, selección...).

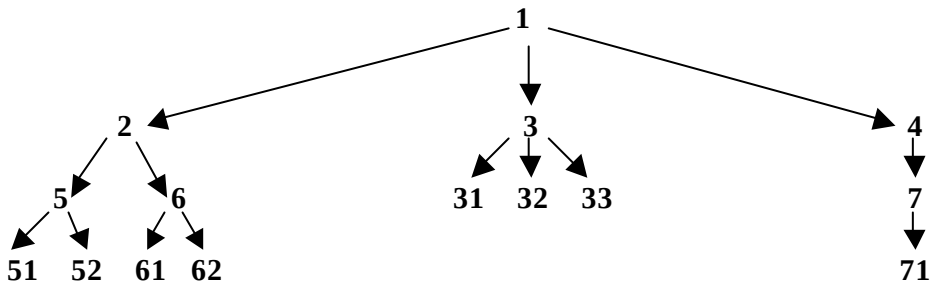
**3.** Programar las funciones primitivas siguientes, para una estructura de datos dinámica **Pila** y también para una **Cola**. Supondremos que cada elemento tiene un entero largo:

- Inicializar estructura.
- Ver si la estructura está vacía.
- Longitud de la estructura.
- Insertar en la estructura.
- Obtener un elemento de la estructura (esto equivale a borrarlo).
- Borrar todos los elementos de la estructura.
- Escribir en pantalla todos los valores que contenga cada elemento de la estructura (en nuestro caso es sólo un entero).

Programar sendos programas para manejar dichas estructuras de datos con todas las opciones indicadas y otras que puedan ejecutarse a partir de ellas (como mostrar todos los elementos de la estructura).

**4.** Deseamos crear una estructura de datos dinámica de tipo árbol, en la que cada nodo almacena un número entero. Además, cada nodo del árbol puede tener un número indeterminado de hijos. Implementar un programa con: (5p.)

- Una estructura de datos para almacenar este tipo de árbol (usando `typedef`).
- Una función `Crea_Nodo_Arbol()` con un único argumento entero que cree un nodo del árbol con el entero que se le pasa por argumentos y devuelva el puntero a ese nodo, sin hijos. Esta función devolverá NULL si se produce algún error.
- Una función `Insert_Hijo()` con dos argumentos: Un nodo del árbol y un entero. La función creará un nodo con el entero y añadirá ese nodo como hijo más a la derecha (último hijo) del nodo que se pasa por argumentos. Esta función devolverá 0 si se produce algún error y 1 en caso contrario. Esta función puede hacer uso de la función del punto anterior.
- El programa principal debe crear una estructura en árbol como la siguiente y en todo momento debe indicar si se produce o no un error:




**5.** Programe primitivas para la estructura del ejercicio anterior para leer un árbol de ese tipo en **preorden, inorden y postorden**.

**6.** Implementar un programa para **gestión de una clase** sin suponer ningún número de alumnos máximo. Para cada alumno se almacenará la siguiente información: Nombre, Apellidos, DNI, Nota en número, y Nota en letra. Para ello se utilizará una estructura de datos dinámica. El programa permitirá las siguientes operaciones, en un menú de opciones:

- Añadir nuevo alumno.
- Borrar alumno ya existente (dando su posición en la lista y/o su nombre, como se prefiera).
- Cambiar algún dato de algún alumno: Modificar sus notas...
- Mostrar todos los datos de todos los alumnos (uno a uno, por ejemplo).
- Cambiar la nota en letra de cada alumno según su nota numérica: No presentado (<0), Suspenso (<5), Aprobado (<7), Notable (<9), Sobresaliente (<10) y Matrícula de Honor (10).
- Calcular la nota media y la varianza de todos los alumnos presentados a examen de la lista, teniendo en cuenta que si la nota es negativa el alumno no se ha presentado y no será tenido en cuenta en los cálculos.
- Mostrar el total de alumnos de cada nota (Suspenso, Aprobado...) y el porcentaje que suponen respecto al total de alumnos en la lista y respecto al total de alumnos presentados (excluyendo los No Presentados).
- Guardar datos en fichero: Esta opción guardará los datos actuales en un fichero (el usuario podrá elegir el nombre del fichero, pero NO la extensión del mismo que será obligatoriamente .DAT).
- Leer los datos de un fichero previamente guardado: Igualmente, el usuario podrá elegir el nombre del fichero del que desea leer los datos.